It is stable and reliable

It has community support,which help us to edit the file

It has strong security

It is cost effective,and open source

We can recode our kernel

Flexible customization

Large community and vendor support

Linux is compatible with many technologies.

It is compatible with cloud environment

Models are secured

Can make career path once you learn linux

As a system administration,

Security and privacy

It is compatible with many new technology

Os is software that enables communication between hardware and software

**What is Linux?**

Linux is an open source operating system.that is freely available with large community support.

**What is an Operating System?**

Operating system is an interface between hardware and end user where the end user can be a human, a robot or it can be any other software. Or we can say an operating system is a software that helps you to communicate between computer hardware and software. operating system is used to utilize system hardware. There are few examples of OS i.e Windows,Linux, Mac,Fedora,Unix,Centos,Ubuntu.



**What do we understand by open source in Linux ?**

In linux we can attach any source code and we have a freedom to modify those code; the codes are openly available to anyone who wants to examine it.This gives you freedom to understand how the software works and identify the bugs and suggest improvement or to contribute in any enhancement to the softwear ,As it is an open source it is typically free of cost and allows you to download install and use without any upfront cost,The licencing under linux runs under open source license such as GNU-GPL(General Public Licence) or it can be handled by or approved by open source initiative(OSI),

**GNU-GPL:**

This widely used free software license ensures that the end user has the freedom to modify, run, share and study the softwear . This ensures that the software remains free and open for all future users .

GPL license software can not be integrated into proprietary software without violating the license

**Open Source Initiative(OSI):**

It is a non profitable organization that protects the open source software by reviewing and approving the definition of open source This allows the user to freely use the software for any purpose . This means the user can modify the source code and distribute those changes to other user.

Linux was developed by Linus Torvalds in 1991, he took this as his hobby project to work on the limitations of UNIX. He used the source code of MiNUX and released the first development under GNU-GPL.

**Types of interface:**

**CLI - Command Line Interface,** this is a command based interface where a user types some command to execute the program.

**Definition:** A text-based interface where users type commands into a terminal or console window to perform specific tasks.

**Examples:**

* Bash in Linux and UNIX systems.
* Command Prompt and PowerShell in Windows.

**Advantages:**

* Offers more control and flexibility for system management tasks.
* Consumes fewer system resources than GUIs.
* Powerful for automating tasks through scripting.

**GUI - Graphical User Interface,** this is an icon based OS where a user can drag or click on the icons.

**Definition:** A visual-based interface that allows users to interact with electronic devices using graphical icons, visual indicators, and menus instead of text-based command labels or text navigation.

**Examples:**

* Operating systems like Windows and macOS use GUIs extensively.
* Software applications like Microsoft Word or Adobe Photoshop.

**Advantages:**

* Intuitive to use, especially for beginners.
* Enables effective management of multiple tasks simultaneously through multi-window and multitasking capabilities.

**Voice User Interface (VUI):**

* **Definition:** Allows users to interact with a system through voice or speech commands.
* **Examples:**
  + Voice-activated assistants like Amazon Alexa, Apple Siri, and Google Assistant.
  + Voice-driven applications in smartphones and smart home devices.
* **Advantages:**
  + Hands-free operation makes it accessible and convenient.
  + Enhances accessibility for users with visual impairments or physical disabilities.
  + Useful in situations where using hands is impractical or unsafe, such as while driving.

**What are the responsibilities of the operating system?**

1. **CPU/ Processor Utilization(CPU Scheduling ):** This determines the order in which process will execute by the cpu. The main goal or primary goal is to get maximum throughput from the processor and the response time should be fast to complete any process.
2. **Process Management:** The main responsibility is to create , schedule , terminate and prioritize any process or threads.
3. **Memory Management:** This involves efficient memory usage , to ensure that the program and the processes have access to the memory whenever they need it .
4. **File Management:**This involves the management of files i.e. organizing, storing, retrieving and manipulating the file and the directories on the storage devices such as Hard Drive, SSD or any other network devices.
5. **Device Management:** This involves control and coordination between any peripheral devices and the computer system. The peripheral devices like mouse, keyword, printer, scanner or any other I/O devices.

**Kernel:**

Kernel is the core part that interacts between hardware and software .This is the first core element that gets loaded in the memory. This kernel communicates between hardware and software. Kernel loads first into memory, when an OS is loaded and remains into memory until OS is shut down. The kernel has a process table that keeps track of all active processes. Kernel is written in C language.

Responsibilities of kernel:

1. **Process Management:**

* Manages the process which include creation, scheduling and the termination.(**in managing the process lifecycle)**
* It ensures that the process gets proper **cpu allocation** to maintain the system stability .
* It manages the CPU resources by deciding which processes get CPU time , in what order and for how long the CPU will be allocated to the process.

1. **Memory Management:**

* The kernel controls the allocation and deallocation of memory.
* This manages Virtual memory, which allows you to use more memory than the physically available memory by swapping the data in and out of the disk.

1. **Device Driver**:

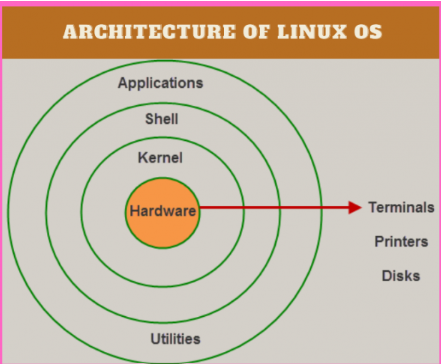
* These are the programs that allow the kernel to interact with the hardware devices like Keyboard, mic, mouse, printer, Scanner or any other I/O devices., to have coordination between the peripheral devices and the CPU.
* It manages the device drivers and ensure that the hardware resource are allocated efficiently

1. **File management:**

* It manages the file system, which organizes and stores the file on various storage devices . The kernel provides functions to create , delete and read and write files and to manage permission and security settings.
* It manages the storage and retrieval of the data on the storage devices.

**The difference between Kernel and OS:**

| **Kernel** | **OS** |
| --- | --- |
| It is a core part of the OS. | It is an interface between hardware & software. |
| It is a software / low level program that manages hardware resources including memory, CPU & I/O devices, interrupt and exceptions. | OS not only includes Kernel but various system utilities, libraries and various user interfaces. |
| It is responsible for doing tasks like process management, memory management , scheduling, hardware management. | Is an interface that helps to interact between user interface. |
| It is a central point of control and resource management in OS. | It controls, manages system configuration files or tools. |



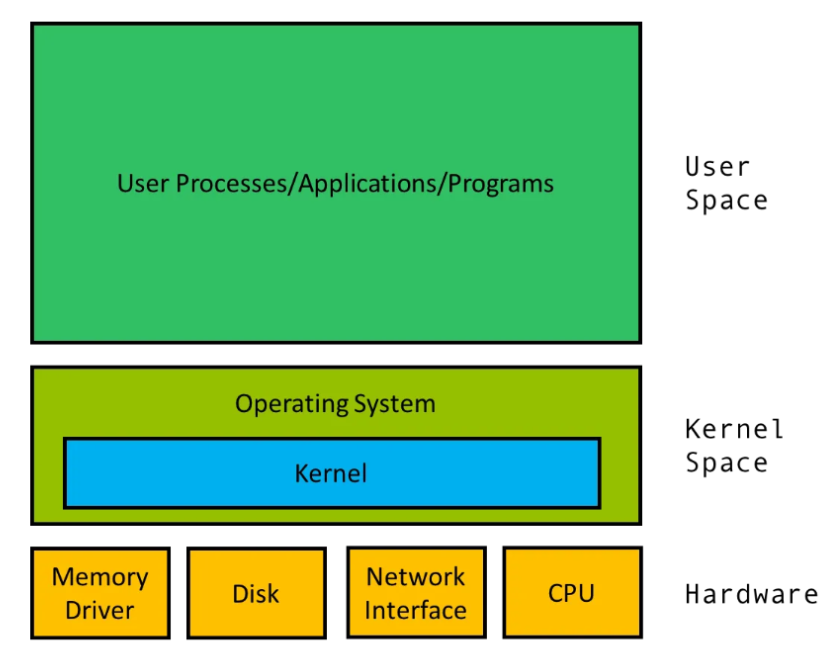
**User Space:** This is the space or region where user applications and processes run.

This space is protected and isolated from the kernel space which means that the code running here cannot directly interact with the hardware or interfere with the kernel operations.

* In this space or environment we run 3rd party or user applications such as web browser , games, ms office application etc.
* Applications in user space have limited access to the system hardware and resources.
* Errors or any crash of the user application does not affect the core component of the operating system .

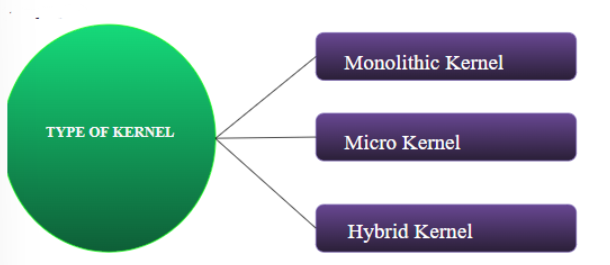
**Kernel Space:** Kernel space or Kernel mode is where the kernel or the core part of the OS operates. This is the restricted area where all hardware and system resources fall.

* The code running in this space can directly interact with the hardware devices and manages the whole system by managing the resources, devices, processes or critical system tasks.
* Kernel space is faster than the user space because there is no need for context switching .
* If kernel Crashes or there is some error in the kernel space it can lead to system failure .



The separation between between 2 spaces i.e user and kernel space is done for the security, system failure or reliability purpose . This is done to ensure that the manfunctioning programs or any malicious software in user space cannot corrupt the kernel or any other running application.

Type of Kernel :



**Monolithic kernel:**

**Monolithic Kernel**

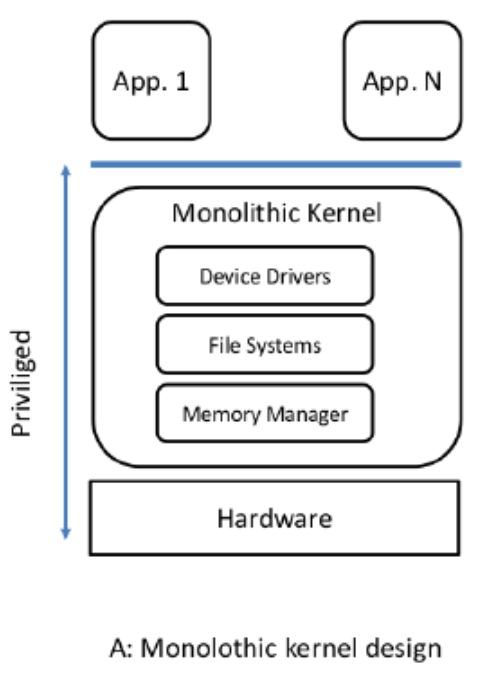
* These types of kernel where all OS services operate in kernel space, this is one of the oldest types of kernel where the entire OS is composed in a single large executable file that runs on kernel mode.
* This is more efficient kernel as it does not have to switch between different address spaces, when executing different tasks. This makes the process execution faster as there is no separate memory space for user and kernel.
* All operating system services run in kernel space, which can lead to better performance but may result in less stability due to the complexity of the kernel.
* Examples: Linux, Unix.

Advantages of Monolithic Kernel

* It is faster because it operates from kernel space.
* It is more secure than any other kernel because it can be protected easily from malicious code.

Disadvantages

* As the entire OS is composed of a single binary executable file that runs on kernel mode so if anything gets corrupted the whole system gets corrupted.

Examples : UNIX, LINUX, Open VMS

2. Micro Kernel:

It is a type of kernel which works between two spaces i.e, user space and kernel space. Micro Kernel works on modularity i.e system services, device drivers run on separate space i.e user space and all the process management, memory management will work under kernel space.

The kernel is minimized, and most services run in user space. This approach can improve stability and security but might incur a performance penalty.

Examples: MINIX, QNX.

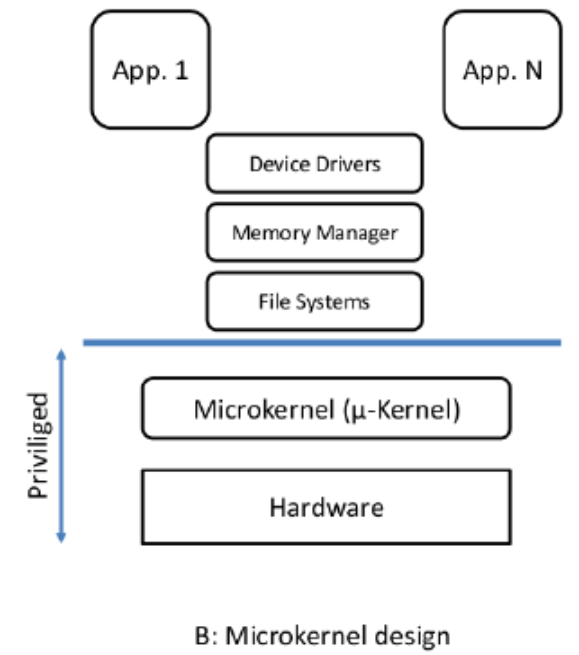
Advantages of Micro-Kernel

It is more stable than Monolithic Kernel, if any services get affected or corrupted it doesn’t affect the whole system.

Examples: Blackberry QNX, MiNIX

Disadvantages

* As it is modular it requires more communication and synchronization between different spaces.
* It uses more system resources such as CPU, Memory than monolithic kernel as it has to maintain synchronization.



3. **Hybrid Kernel**

This is a combination of monolithic and micro kernel, it has speed of monolithic kernel while it has modularity and stability of micro kernel. All the essential services like process management, file management, CPU management, CPU scheduling, interrupts are kept in Kernel mode while services like device drivers are placed under User mode.

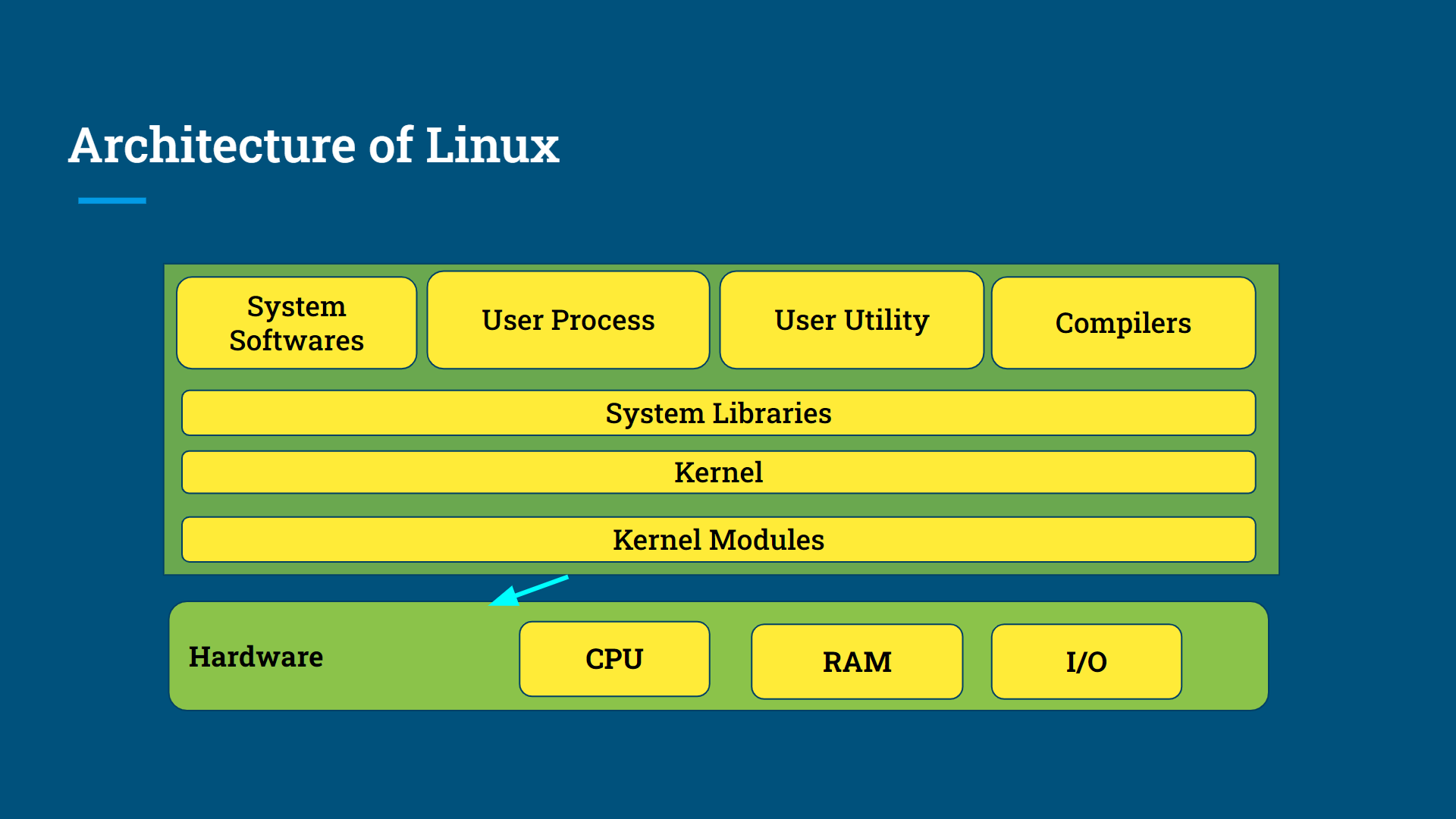
Combines aspects of both monolithic and microkernel architectures, trying to balance performance and modularity.

Examples: Windows NT, macOS.

Examples : MS Windows, MAC OS, Netware.

**Difference between Micro and Monolithic Kernel**

| Micro Kernel | Monolithic Kernel |
| --- | --- |
| User services and kernel services are kept in separate spaces | Both kernel and user services are kept in same space |
| Smaller than Monolithic Kernel | They are larger than micro-kernel |
| It is easier to add new functionality as it works in modularity | It is difficult to add new functionality as everything is there in one space |
| Failure of one component doesn’t affect the whole system | Failure of any one component affects the whole system |
| Execution speed is slower | Execution speed is faster |
| Ex : Blackberry QNX | Ex : UNIX |



## Hardware

CPU - It is a central processing unit which is responsible for executing and controlling any process or task on the system.

RAM - It is said as Random Access Memory that helps to store data temporarily

I/O Devices - Input devices are those devices which help you to enter data or commands into the system for example - Keyboard, Mouse, Scanner.   
Output devices are those devices which help you to display the output from the system for example - Monitor, Printer, Speaker

## Kernel Modules

It is also known as device driver or also referred to as loadable kernel .

It is called as loadable and unloadable kernel because a piece of code can dynamically be loaded into linux kernel without rebooting your system. These are basically written into C / Assembly language

Examples - Graphic Card, Network Adapter.

Kernel modules are stored in files with extension of ‘.ko’ and are loaded into kernel using ‘INSMOD’ or using ‘MODPROBE’ commands.

The main subdirectory for kernel modules is found under ‘/lib/modules’.

## Kernel

It is a core part of the OS that exists between the hardware and software, whenever a system boots kernel is the first part that gets memory and exists till the system shuts down. It is responsible for managing CPU memory, I/O devices that handles system calls, interrupts and any exceptions.

## System Libraries

They are prewritten codes that provide functionality to programs and applications running on OS. It is a set of functions for user level applications to interact with OS Kernel.   
Examples - C Standard Library (LibC) this library is used to perform operations like I/O, Memory management and string manipulation,

Math Library (LibM) provides a wide range of mathematical functions that includes Trigonometry, Logs.

Database Connectivity (Libpq, LibMySQLClient) These libraries are used to interact with various databases on your system.

System Call Wrappers this library is used to make system call to interact between user level program and kernel

Example - printf

## User Utility

It provides various functionality tools to user and system administrator to interact and perform various tasks on Linux   
Example - Shell (Bash, CSH, ZSH), Shell command is used to interact with system by entering some commands

Text Editor this is used to create and edit a text file.   
Example - Vim, Nano, Touch

File & Directory Management, this helps you to manage file and directories including managing files by commands like cp, rm, ls, mkdir

Process Management allows you to view and manage the process with commands like ps,kill,top.

Package Manager this helps you to install and update system software, Examples - apt, yum, pacman.

File Compression this helps you to compress and decompress any file. Examples - Zip, Unzip, Tar

Remote Desktop this utility allows you to remotely access any other system Examples - SSH PUTTY

Networking Tools this helps you check the network using ping,ss commands

## User Processes

These are those programs that run in user space which are protected and isolated areas of memory and execute in separate space. They are initiated by user to interact with kernel and system resource through system call following are the different modules through which user processes work :

* User Application - User processes consist of user application which can either be graphical application or command line utility to interact with the system.
* User Space - User processes that operate within user space, it is a protected environment where processes are given limited and direct access to hardware resources.
* Process Creation - Processes are created by users from another parent process fork() ,exec() are system calls that are used to create and replace any processes.
* Resource Management - Processes interact with kernel to manage various resources like CPU Memory, I/O.
* System Calls - User processes interact with the kernel by making system calls. These calls request services / operation from the kernel, few system calls are like creating new processes, allocating memory, doing I/O operations, creating new files and reading files.

## System Software

It refers to collection of software and utilities that are required to manage operation on OS

Examples -

* Init - System is responsible for initializing, managing system services and handling system shutdown process, this is responsible for initializing OS after it loads into memory.
* Bootloader - This is responsible for loading OS into memory, it is the first program that is executed whenever you turn on the system.

## Shell

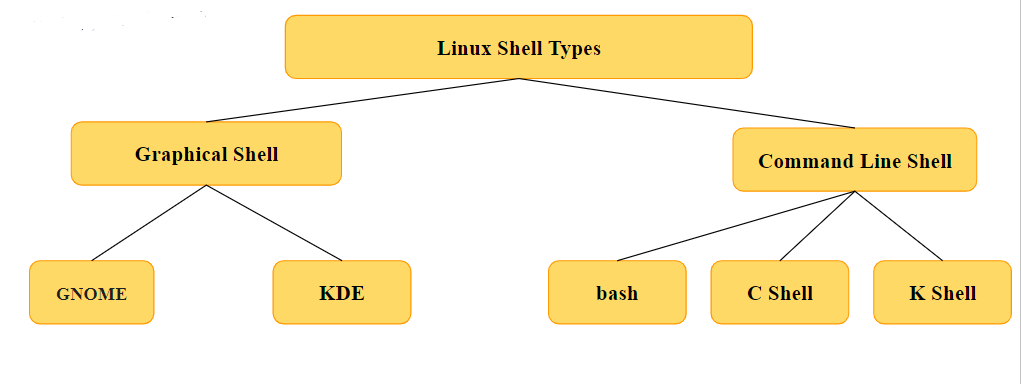
It is a command line interface that allows you to interact with kernel and user. It is a program that interprets user input and executes the commands or we can say it is a program that executes any other program.

Following are the tasks that are performed by shell :

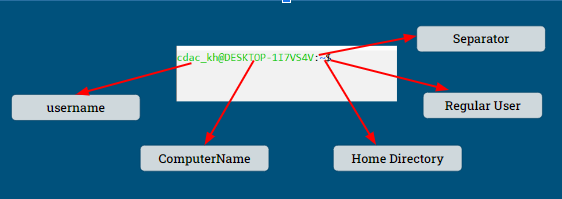
* User can execute the command
* Shell provides I/O redirection which allows the user to redirect I/O from the command to and fro from the file.
* This manages system and environment variables using shell ,users can modify system variables.
* Shell allows the user to write a script that can be executed as a single unit.(Script is a collection of commands).

Shell is divided into two parts

* Graphical - This shell specifies the manipulation of a program using a graphical interface that provides operations like moving, closing, resizing or switching between applications.
* Command Line - It is a program that provides command line interface for interacting with OS. It allows users to enter any command on prompt and executing.



Linux Command:



‘:’ This is a separator

‘~’ It shows the user working directories, if we change the directory the sign will vanish

‘$’ Suggests that you’re working as a regular user

‘#’ this mean you’re working as root user

‘/’ this sign represents the sign of root

Username can be anything you have given to your system

Hostname / Computername this helps to identify computer over any network

Non Root user default prompt is ‘$’

Default prompt for root is ‘#’

## Path

It is a unique location to a file / folder in a file system. A path of a file can be a combination of ‘/’ and alphanumeric characters.

Example : /lib/module/file.txt (where lib is a parent directory and module is a sub directory inside lib and file.txt is file inside module)

There are two types of path

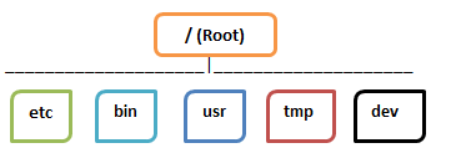
* Absolute Path - This specify location of file and directory from root directory in other way we can say absolute is a complete path from start of actual file system from root(/) directory.  
  Example - /lib/module/file.txt
* Relative Path - It is defined as path related to present working directory, it starts from your current directory   
  Example - Suppose your file is located at ‘ /home/usr/document/ ‘, your file is located at document directory in relative path it will be referred as ‘ /document/file.txt’

## Users in Linux

User is referred as individual who is interacting with the OS to perform various tasks, there are 3 types of users

* Regular User: These are those users which are created during installation of Linux in this all the files and folders are stored in /home/ directory and they can not have access to directories of any other user. Their symbols is ‘$’ and can also be ‘%’
* Root User: They are the super users that have all the admin privileges. Its symbol is ‘#’
* Service User: This is widely used as server OS and services like e-mail and other applications have their own services account.

## File system in Linux



The files in Linux are stored in tree structure where root is considered as start of the file system . This file system is further divided into 3 parts

****

* General - It is also referred to as ordinary files they contain images,text, config files they can store files in ASCII or Binary format. These are the most commonly used file types by linux user.
* Directories - These are special types of files that contain directories and files.
* Device - These are special files that represent physical and virtual devices in the system such as printer, Hard drive, CD-ROM.

Directories

* /boot - This contains bootloader and kernel files, it also contains all the files that are needed during the boot of the system.
* /home - This contains all the program files that are created by the user. It also contains user data, config files, user documents (all the personal files of a user).
* /bin(Binary Binaries) - This contains all the binary executable files, it also contains all the user commands in binary format like cd, pwd, mv, rm.
* /usr - It is also referred as Unix System Resources this contains all the user binaries, documentations, libraries, installed softwares and read only program data
* /etc - It is referred to as Editable Text Configuration that contains all the config files which are used by system services; it also contains system startup and system shutdown script that is used to start any program.
* /var - It is referred to as Variables, contains all the variable data like log files generated by the system, cache files generated for applications.
* /tmp - It is referred to as temporary, contains temporary files created by various applications typically these are cleared when the system reboots.

## Difference between Windows and Linux

| Windows | Linux |
| --- | --- |
| Windows is Closed Source | Linux is Open Source |
| Windows has different Data Drivers | Linux has a tree like structure |
| Peripheral Devices in windows are like printer, CD-ROM | While here they are considered as files |
| There are 5 different kinds of users | There are 3 different kinds of users |

1. **System Information**: As a system administrator you need to know about the exact version of kernel or what operating system is there, this help will help you to identify the software that is compatible with your kernel and hardware .

Whenever you are updating your system you need to know the current environment or precise details about your OS.

* **Uname : This is used to check OS**
* **Uname -r : Is used to check the version of the kernel.(identify the kernel running on our system )**

6.8.0-41-generic

6.8.0 : main version

41- patch level

Generic : this indicates the specific configuration or build type - Ubuntu

* **Uname -v: Is used to display OS/Kernel versions.(know the built date and time and other metadata)**  
  #31-Ubuntu SMP PREEMPT\_DYNAMIC SAT APR 20 00:40:06 UTC 2024
* **#31-Ubuntu SMP PREEMPT\_DYNAMIC :** this means that this is a symmetric multiprocessing system with preempt support.
* **SAT APR 20 00:40:06 UTC 2024 :** shows the date and time of kernel built.

**Where do we use this:**

1. Updation: as system administrator to determine which patch or update to be used
2. Software compatibility: developers often need to know the kernel release that is compatible with our software .

**Uname -a:** Is used to display all the system info.

This is used to display system information. The output typically includes:

* The kernel name.
* The network node hostname.
* The kernel release.
* The kernel version.
* The machine hardware name (architecture).
* The processor type (if available).
* The hardware platform (if available).
* The operating system.

**Uname -s:** Is used to check OS same as uname

**Uname -m:** Is used to display machine hardware.

**Host Name: It gives you the hostname of your system**  
Hostname -I: It gives you the IP address of your system  
Hostname -f: It gives you the qualified domain name

How to set new Hostname: hostnamectl set-hostname newhostname

**Whoami** is a command that is used to display the current user ID and username of the user who is currently logged in, this will print the username of the user who is running the command. This command is useful when you want to check the currently logged in user onto the system.

**Date: It is a command that is used to display Date and Time.**

* **date:** This will give you current date and time
* **date +%y-%m-%D:** This displays date in YYYY-MM-DD format.
* **date +"%H:%M:%S":** Thisdisplays the current time in the format 12:34:56 (Hour:Minute:Seconds).
* **date +’%A’ :** This displays the current day.
* **date -u:** This will display the date according to the UTC Time zone.
* **date -u +’%Y-%M-%D %H:%M:%S’:** This displays utc time in given format  
  **date +%T:** This will display time in HHMMSS format.
* **date +"%A, %B %d, %Y":**Displays a more readable format like Monday, September 04, 2024

**Useful Format Specifiers**

* %Y - Year (e.g., 2023)
* %m - Month (01 to 12)
* %d - Day of the month (01 to 31)
* %H - Hour (00 to 23)
* %M - Minute (00 to 59)
* %S - Second (00 to 59)
* %A - Weekday name (e.g., Monday)
* %B - Full month name (e.g., September)

date MMDDhhmmccYY.ss: Where MM denotes Month, DD denotes Date, hh denotes hours, mm denotes minutes, cc denotes century, YY denotes year and .ss are the seconds(this is optional)- This command is used to set new date and time for the system.

**sudo date +%Y%m%d%H%M.%S -s ‘new date and time’: This command is used to set a new date and time.**  
  
**Why to learn this command or where we can use this command 👍**

1. When you want to automate any task for a particular date and time.
2. It is useful in creating logs, backup .
3. When we want to grant or revoke some permission
4. Whenever you want to ping any other server or establish a connection over the network you need this command to be checked.
5. The date command can be paired with touch to modify the timestamp of the file , this is important for scripts that are dependent on the file modification time .

**cal/ncal: Is used to display calendar of particular year**

cal -y: It displays calendar for the year entered   
cal -3: It displays the previous month, current month and next month  
cal -3 m y: It displays calendar for specific month of the year along with the previous and coming month  
cal m y: It displays calendar for particular month in a year

**Where We can use this cal command:**

1. Report generation :
2. Meeting Scheduling
3. Administrators can use to know the current date to cross verify any logs or records.

**Uptime:** Is used to display how long a system has been running, along with other information such as current time, number of users logged in, system load average. This command is generally used for diagnosing system performance

Uptime   
12:34:56 - Current time

up 10 days, 3:48 - System has been running for 10 days and 3 hours 48 minutes

2 users - Number of users currently logged in

load average: 0.01, 0.05, 0.05 - Load averages for the last 1, 5, and 15 minutes

**When we can use this command:**

1. To identify issues after a crash, if the system shuts down or crashes , then uptime can help us to determine if the system has recently restarted.
2. To check the load(how busy our system is) and the number of the user logged on to the system.
3. To analyze the system performance and further we can plan the capacity of the system .
4. As a system administrator, we can use this command to verify how long a server or a system has been running.

**Users:** It is used to display the list of users currently logged in to the system.

**w:** It is a command used to display the information about currently logged in users, this includes the information related to username, log in time, idle time, system load and what operation they are performing.

* **USER:** The username of the logged-in user.
* **TTY:** The terminal type that the user is logged in from.
* **FROM:** The hostname or IP address from where the user is accessing the system.
* **LOGIN@:** The time when the user logged in.
* **IDLE:** The idle time of the terminal, i.e., how long since any input has been detected on that terminal.
* **JCPU:** The time used by all processes attached to the tty.
* **PCPU:** The time used by the current process, named in the WHAT field.
* **WHAT:** The command line of the process that the user is currently running.

**who:** This displays information about users who have logged in with their username, terminal session and their log in time.

**NAME:** The username of the user logged into the session.

**LINE:** The terminal or tty (teletypewriter) from which the user is logged in.

**TIME:** The time when the user logged in.

**COMMENT:** Often shows the hostname or IP address from where the user is logging in.

**who -q :** Displays only the names and number of users currently logged in.

**who -r :** Shows the current runlevel of the system.

**who -a :** Displays all information, combining multiple other options for a comprehensive view.

**who -b :** Shows the last system boot time.

**who -H :** Includes column headers in the output.

**who -u :** Shows the user list with idle time, adding an idle column to the standard display.

**Why do we need the command?**

1. To know how many users and which user has logged in to the system .
2. How many sessions each user has opened and their login time .
3. To ensure the authorized users are accessing the system .
4. To know the time of system boot .

**last**: This displays information about the last logged in user with their username, log in time, system shutdown, reboot time and terminal session.  
 **USER:** The username of the user.

**TTY:** The terminal line from which the user logged in.

**IP ADDRESS/ HOSTNAME:** The remote host IP address or hostname from which the user accessed the system (if the user logged in remotely).

**LOGIN TIME:** The start time of the user's session.

**LOGOUT TIME:** The time when the user session ended, or the duration of the session if the user is still logged in.

**DURATION:** The total time the user was logged in during that session.

**Why do we need this command?**

1. To identify unauthorized or authorized users with their user logins and logout.
2. We can generate a log report of all the last logged users

**man:** It is used to display manual page for any command, manual page contains information about the command, options that can be used with the commands along with other information.

**Whatis : whatis “command” :** this gives you one line description of any command , this can be used to quickly review or get any reference for the command.

Why do we need to learn this command?

1. For quick reference/ description for any command.
2. Easy to understand what a command will do.

Top:It is used to display the information related to the process that are running on the system.

**PID:** Process ID.

**USER:** User owning the process.

**PR:** Priority of the process.

**NI:** The nice value of the process which affects its priority.

**VIRT:** Virtual memory used by the process.

**RES:** Resident memory used by the process.

**SHR:** Shared memory of the process.

**S:** Process status (sleeping, running, stopped, etc.).

**%CPU:** Percentage of the CPU used by the process.

**%MEM:** Percentage of physical memory used.

**TIME+:** Total CPU time used since the process started.

**COMMAND:** The command that launched the process.

**top -u ‘username’: This will display you the information related to the specific user.**

top -d “10” : this will give you an update of processes that are running every 10 seconds.

Top -i: display you the active processes .

**ps:** This displays the information about the current running processes on the system .

ps -A: this will give you all the processes that are running on your system.

ps -u username: Thi shows the processes that are running on the specific user .

ps -f: This give the full format listing which include :

UID:User ID of the user that owns the process

PID: Process ID

PPID: Parent Process ID

C: CPU utilization

STIME: Start Time of the process

TTY: Terminal type associated with the process

TIME: Cumulative CPU tIme.

CMD: Command name.

**ps -ejH/ ps axjf**:This gives a hierarchical display of the running process . This is the visual representation of processes and their parent and child relationship .

PID: Process IDPS

PGID: Process Group ID(This is the ID of the process group to which a process belongs to.The process belonging to the same group can easily communicate to other processes in the same group.)

SID: Session ID: this is used to identify the session in which the process is running.

TTY: Terminal type associated with the process

TIME: Cumulative CPU tIme.

CMD: Command name.

**ps aux:** This command is used to view all the processes that is running on your system along with detailed information like

User: User are those username who own the process

PID: Process ID(Unique identification of any process)

%CPU: The percentage of CPU utilization or percentage of CPU time utilization.

%MEM: This represents the percentage of memory utilization by the current process.

VSZ: (VIRTUAL MEMORY SIZE): The total amount of the virtual memory used by the process.(representation is in KB).

RSS: Resident Set size: This shows how much RAM a process is consuming

TTY:Terminal Type. If any process is not attached to any terminal then in the terminal section you will get?.

STAT(State): This is the state of process that is running on the System

R: Running

S: Sleeping

I:Idle

T: Stopped

Z: Zombies

START: Start Time of the process

TIME: Cumulative CPU tIme.

CMD: Command name.

ps -e:It displays information about all the processes (including all the other users)

**Why need to learn this command:**

1. We can monitor CPU, MEMORY, VIRTUAL memory usage.
2. We can troubleshoot the processes that may be in idle state or in Zombie state.
3. Security Purpose: We can identify any suspicious processes that are running on our system.

**KILL:**

**kill:** It is used to terminate a process by sending signal to them   
**kill ‘process id’:** This will kill the particular process.  
**Kill -9 ‘pid’:** This will kill the process forcefully.

**free:** It is used to display information about the system that is how much memory is available, allocated   
free -h: Gives you the info in human readable format (The sizes are specified in MB,KB)

**Free -m** : Shows the memory usages in megabytes.

**Free -g** : Shows the memory usages in gigabytes.

**Free -t :** show all the total of RAM and Swap

**Free - s 2:** continuously update the display at an interval of second specified in my case it is 2 Second.

**Why do we need to know this command?**

1. Monitoring the memory usage , how much memory has been used and how much memory is left.
2. System better performance: You can distribute the memory for better system performance.

**ping ‘IP’:** It is used to send request packets to specific IP addresses to check whether the target IP / Host is reachable or not.

**Ping -c count google.com**

With option c: stops after sending or receiving count the packets received

**Why do we need to know this command?**

1. To diagnose the network connectivity issue between the 2 servers i.e. client and destination server.
2. To verify the network connection of any server , whether it is functioning properly or not .

**df:** It is used to display disk spaces used by filesystem  
**df -k:** It will display disk spaces used in 1KB blocks   
**df -m:** It will display disk spaces used in 1MB blocks  
**df -h:** It will display disk spaces used in 1GB blocks   
**df -T:** It will display the types of file systems

**du:** It is a command that is used to display the uses by file and directory, this provides the information of the disk spaces used by file and directory in specified location  
du -h ‘directory name’: It will display the directory size in MB,KB and GB

**Why do we need this command?**

1. To determine the free space and consume the disk space wisely(by removing the older files or no longer used files).
2. To optimize the disk uses.

**ifconfig:** It will display configuration related to network interfaces  
**ip addr show:** It will display the IP address and the information related to network interface

## **Accessing File System**

1. **pwd:** It is used to display the present working directory. It is also known as print working directory, this gives you the absolute path of the current working directory.

**Why do we need this command?**

1. It can be used to navigate and to know the current directory where the user is working.
2. To know the relative path.

**/bin/’any command’ –version:** This will display the version of any command.

1. To check the compatibility while writing the script, it is often useful to check the version of the critical commands
2. To identify the bug or change in working , we need to know the exact version.

mkdir ‘directory name’: This command is used to create new directories

mkdir -p t1/t2/t3 : This will create a nested directory.

Mkdir -p new/{n1,n2}:

new/

├── n2/

└── n3/

Mkdir -p new1/n1 new2/n2

1. To organize the file/directory into hierarchical structure which is to manage files in more logical ways

2. Data segregation: we can segregate the data and make it for better data management.

Cd: This command is used to change the directory ,it can also be used to move to the home directory .

1. cd ‘directory name’: This command is used to change the current directory
2. Cd .. : This command will move to the previous working directory.
3. Cd -: Will print last working directory
4. Cd ~: Will move to home working directory
5. Cd /: Will move to the system working directory.
6. Cd ~’user’: Will move to the user directory.

## Creating Files in Linux

1. touch: It is used to create empty file and update the access, modifying the timestamp of the file. It can create multiple files using touch ‘file1’ ‘file2’ ‘file3’...  
   If the file doesn't exist then touch will create a new file with new timestamps and it will be updated
2. Echo: It is used to print and display the content on the terminal when it is used with the redirect operator (>) it will create a file adding text to the file.  
   Syntax : Echo “Hello World” > file1.txt   
     
   If you want to add content to an existing file without overriding the content then we will use ‘>>’ operator

If you want to add content to an existing file without overriding the content then we will use ‘>>’ operator  
Syntax : Echo “Hello World” >> file1.txt

1. Printf: It is similar to echo which creates new file and updates pre-existing files   
   Syntax : printf “Hello World” > file1.txt  
   Similar Syntax if you do not want to override  
   In printf we can use escape characters while in echo we can not use escape characters.

printf "Hello world\n" >> f1.txt

printf "Hello world\t" >> f1.txt

1. Nano: It is used to create files and its syntax is : nano ‘filename’ , to save the file we press ‘Ctrl + o’.  
   Exit - ‘Ctrl +x’   
   Cut / Delete from current line - ‘Ctrl + k’  
   To Search - ‘Ctrl + w’  
   To Paste / Uncut - ‘Ctrl + u’
2. Vi Editor : It is used to perform tasks like creating, editing, saving and navigation  
   To insert - Press ‘i’  
   To move to Escape mode: Press ’Esc’  
   Move to beginning of the file in escape mode: Press ‘gg’ or ‘/G’  
   Move cursor to left: ‘h’  
   Move cursor to right: ‘l’  
   Move cursor up : ‘j’  
   Move cursor down: ‘k’  
   Search forward : ‘/search’  
   Search Backwork: ‘?search’  
   Delete: ‘dd’  
   Undo: ‘u’  
   Redo: ‘Ctrl + r’  
   Copy: ‘yy’  
   Paste: ‘p’  
   Quit: ‘:q!’  
   Write and Quit: ‘:wq’  
     
   Normal Mode: For navigating and editing text, where keyboard inputs perform editing and navigation functions.

### Insert Mode: For inserting text. Accessed from Normal Mode by pressing i, a, o, etc.

ESC Mode: Esc mode refers to the mode you enter when you press the Esc (Escape) key

* **Entering Insert Mode:**
  + i: Insert before the cursor.
  + a: Append after the cursor.
  + o: Open a new line below the current line.
* **Exiting Insert Mode:**
  + Esc: Return to Normal Mode.
* **Saving and Exiting:**
  + :w: Save the file but keep it open.
  + :wq or ZZ: Save the file and quit vi.
  + :q: Quit if there are no changes.
  + :q!: Quit and ignore any changes.
* **Cut, Copy, and Paste:**
  + dd: Cut (delete) a line.
  + yy: Copy (yank) a line.
  + p: Paste below the cursor.
  + P: Paste above the cursor.
* **Undo and Redo:**
  + u: Undo the last operation.
  + Ctrl + r: Redo the last undo.
* **Moving Around:**
  + h, j, k, l: Move left, down, up, right, respectively.
  + 0: Move to the beginning of the line.
  + $: Move to the end of the line.
  + G: Move to the end of the file.
  + gg: Move to the beginning of the file.

**Search and Replace with Confirmation:**

* **Command:** :%s/old/new/gc

**Autocompletion:**

* **Command:** Ctrl-n and Ctrl-p (in insert mode)

**Faster Scrolling:**

* **Command:** Ctrl-u and Ctrl-d

**Editing Multiple Files:**

* **Command:** :args file1 file2 file3 then use :next, :prev, :first, :last to navigate.

**Diff and Merge:**

* **Command:** vimdiff file1 file2

**Block Selection and Editing:**

* **Visual Block Mode:** Press Ctrl-v to enter visual block mode.

**Nano Vs Vi:**

| nano | Vi |
| --- | --- |
| 1. graphical/command based interface   Which made this a user friendly | 1. It is bit complex because as a beginner you need to know all the modes in which VI editor works |
| 1. Easy to learn as shortcuts are displayed | 2. Need to memorize each commands |
| 1. It is bit less efficient then vi editor | 3. It is good to handle large file and can do complex editing |

1. ls: It is used to show the list of files and directories  
   ls -l: This shows the files modification size, time, file, folder name and their owners with the permissions  
   ls -a: This will list out all the hidden files (starts with . and ..)  
   ls -it: This shows the list of files and directories with modified date in ascending order  
   ls -S: This shows list of files and directories in descending order  
   ls -n: This gives the user, username, group id of any file or directory  
   ls ~: This gives you the list of files and directories in home  
   ls \*: This gives you list of directories and sub directories  
   ls -i: This gives you inode of the directories.(inode is index node that is a unique number which holds metadata of file, attributes of file like size, owner of file, creation of file, permission, location, file type and any other link attached to the file)  
   ls -R: This will list you all the directories in tree format.  
   ls -r: This will show you files and directories in reverse order.  
   ls -l /temp: This gives you the list of temporary files.  
   ls -IS: This gives you a list of files and directories in order of their sizes.  
   ls -d \*/: This gives you a list of directories.

rm ‘filename’: It is used to remove directories, files and content within the directories   
rm ‘filename’: This removes the specific file  
rmdir ‘directoryname’: Removes the specific directory name  
rm -r ‘directoryname’ : This deletes the directory along with its contents

cp : It is used to copy files and directories from one location to another  
Syntax - cp ‘source’ ‘destination’   
cp -n ‘source’ ‘destination’ does not override already existing file  
cp -r ‘source’ ‘destination’ copies directories from one place to another  
cp -a ‘source’ ‘destination’ preserves the file attributes.

**. (Dot) - Current Directory**

The single dot . represents the current directory. It is a reference to the directory that the user is currently in.

### **.. (Dot Dot) - Parent Directory**

The double dot .. represents the parent directory of the current directory. This is the directory one level up in the hierarchy.

1. mv: It is used to move file from one location to another   
   Syntax - mv ‘source’ ‘destination’  
   mv ‘file.txt’ ‘destination’ - This is moving file to a destination  
   Mv ‘file.txt’ ‘destination/new.txt’ - This command moves and renames the file   
   mv dir1 ‘destination’ - This moves the directory  
   Difference between CP and MV command

| CP | MV |
| --- | --- |
| It is used to copy a file and directory | It is used to move a file or directory from one location to another |
| This works as copy and paste | This works as cut and paste |
| It cannot be used | MV command can be used to rename a file |
| The file attributes changes as we copy | The file attributes remains the same even after renaming |

rsync: It is a command that is used to transfer and synchronize files and directories between two networks, these are specially used to keep backups or to remotely synchronize files.  
Rsync gives   
  
Syntax: rsync ‘source’ ‘destination’  
Rsync -av ‘source’ ‘destination’ : Sync a directory locally  
Rsync -av -e ssh ‘source’ user@remotehost:’destination’:

**Difference between rsync and mv**

| rsync | Mv   |  | | --- | | It overrides the existing file | |
| --- | --- | --- | --- |
| 1. Used to copy and synchronize the file or directories | It is used to move or rename any file or directory |
| 1. It can transfer data either locally or remotely | 2. It copy or rename locally |
| 1. It also provides detail information about file transfer in , file sent and received | 3. It doesn't give full information |
| You can take backup as well |  |
| It copies the difference and update it |  |

Rename:

We can rename a file with using regular expression:

Regular Expression: It is also known as regex ie the sequence of characters thatDefine a search pattern which can be used for manipulating the text.This can be used for replacing, splitting and for validation of any text.

Components:

1. Literals: rae the matching characters that will be matched in the string
2. Metadata: Symbols that hold some meaning

. –: match any single character

^ –: match the start of the string

$ –: match end of the string

“ \* ”: match zero or any or more element

“+”: one or more elements.

[]: A set of character to match any single character

Eg: [0-5]: this will try to match the value from 0 to 5

[a-z]: this will match all the character that is from a to z

[A-Za-z]: This will match all the alphabets in both the cases ie upper and lower

| : this is logical or

Eg: a | b : either a or b

() : This allow you to apply regex operators to the entries grouped pattern

{}: This is used to match in more precise

? : match zero or one occurrence

**\ .** : These are used with escape character and are generally used with the symbol metadata

s: substitution operation

y: to change the character ranges

/: act as a separator

^Hello$:

| rename | Mv |
| --- | --- |
| 1. Rename the file using regular expression | 1. This does not uses regular expression |
| 1. This renames multiple files or rename in bulk | 2. We only rename one file at a time |
| It does not changes the location of the file | 1. It change the location of the file |

wc: It is used to count the number of words,lines and characters in a file  
Syntax : wc ‘filename’: Displays the output as number of words,line and characters in a file  
wc -l: Returns number of lines   
Wc -w: Returns number of words  
Wc -m: Returns number of characters   
Wc -L: Returns the longest line  
wc -l \*.txt :Count Lines in Multiple Files

cat file.txt | wc -l: command sequence counts the number of lines in file.txt.

Grep: It stands for Global Expression Regular Print, this command checks for specified patterns in the entire file. This means it will check the entire file for a particular specified pattern passed to it.  
Syntax: grep -n ‘pattern’ ‘filename’: Returns number of lines where pattern has been found   
grep -c ‘Hello’ ‘filename’: Returns number of patterns matched.  
grep -v ‘Hello’ ‘filename’ : Will display number of lines where patterns wasn’t match  
grep -e ‘Hello’ ‘filename’: Will display matches in case sensitive manner  
grep -i ‘Hello’ ‘filename’: Will display matches in case insensitive manner  
grep -r ‘Hello’ ‘filename’: Will find for matches recursively in directories and subdirectories  
grep -o ‘Hello’ ‘filename’: Will display matches without the lines.  
grep -w "hello" filename.txt: Matches only lines where "hello" stands as a whole word, not as a part of another word.

grep -n "Hello" filename: Displays the matching lines and their line numbers.

grep "search\_string" file1.txt file2.txt:Searches for "search\_string" across multiple files.

-B (before), -A (after), -C (context)

grep -A 3 "search\_string" filename.txt shows 3 lines after each match.

grep -B 2 "search\_string" filename.txt shows 2 lines before each match.

grep -C 1 "search\_string" filename.txt shows 1 line before and after each match.

grep “^H” f1.txt: Show all that matches H at first character

1. find(imp) : It is used to search files and directories within specified directory hierarchy, it searches for files on criteria such as name, size, permission, timestamp  
   Syntax : find ‘path’ -’name’ ‘filename’, this finds file with specific name.

find . : all files and directories in current directory  
find ‘path’ -type d, this will find directories.   
find ‘path’ -type f, this will find files  
find /’filename’, this will search file in the entire system   
find ~’filename’: This will search in home directory  
find. ‘filename’: This will search in current directory  
Examples :   
find /home -type f -mtime -10, find in home directory as per modifications in last 10 days.  
find /home -type f -perm -777, find in home directory as per read,write and execute permission.  
find /home -type f -size +1k, find in home directory as per size of files exceeding 1KB.  
find /home -type d -empty, find in home directory for directories that are empty.  
find / -size +100M, find files larger than 100MB

find / -atime -1, find files accessed in the last 24 hours

find path -type f -name "\*.txt" -mtime -1: will show on the specified path, file extension txt that have been modified in last 1 day

| find | grep |
| --- | --- |
| 1. It is used to find the file based on different attributes like size, permission, types, modification time , access time | 1. It is used to find pattern within the file content |
| 1. We don't use regular expression with find | 2.We use regular expression to find the pattern |

Locate: This command as the name suggests will locate any file / directory  
Syntax: Locate ‘filename’/‘directoryname’ :

Locate -r ‘‘directoryname’, It is used to locate directories

Locate -u ‘username’, It is used to locate files and directories owned by specified user

Locate -e ‘filename’, It will locate all the update files

locate -i filename : -i option makes the search case insensitive

locate -n 10 filename : n option limits the output to the specified number of results. In this case, it will show only the first 10 matches.

locate --regex 'filename$' : --regex option allows you to use regular expressions to refine your search. This example finds files that end with "filename

| Locate | Find |
| --- | --- |
| 1. It used to search file or directory based on location | 1.It is used to search file name based on files attribute or metadata |
| It is faster than find command | 2. It is slower than locate |
| It is able to find the files even after they are deleted | 3. Find cannot search the deleted files |

Link:

Links : Link refers as reference to files and directories there are two kinds of links we can create  
A. Hardlink: It points to an existing file in the filesystem and creates a copy, in this kind of link if the main file is deleted the hard link remains the same(content and location).   
Syntax: ln ‘existing file’ ‘newfile’   
It is efficient in creating backups it is also helpful in sharing configuration processes   
  
B. Softlink: It is also referred to as symbolic link in this pre-existing files or directories points to symbolic link. In case pre existing files get deleted the softlink will also be deleted  
Syntax: ln -s ‘existing file’ ‘newfile’   
Softlink is used to create shortcuts for files and directories, that helps us to share the common resources.

## 

## 

## 

In Linux , File authorization is divided into 2 parts:

1. Permission
2. Ownership

Linux File Ownership: This gives the information about file/directory owner(this means file belongs to which owner or group).

This is further divided into 3 parts:

1. User
2. Groups
3. Others

Linux file authorisation is divided into 2 parts :

1. Permission
2. Ownership

**Users:**

Users are those who own the file . By default those who create the file are the owner.These owners holds some attribute like

Id: this will print user id, group id, others id and these id have some predefined meaning.

I.e 0 means the ownership is with root and the 0 has been reserved for the root.

**UserID:** this is also called as user identity or uid , that is assigned to the user to identify the type of the users and to understand the system resources utilizations.

There are range for these id:

0: reserved for root

1-99:reserved for predefined accounts

100-999: reserved for system administrator and accounts

1000-10000: reserved for application accounts.

Above 10000: reserved for user accounts.  
We can change user permission using symbolic characters too   
Syntax : chmod u=rx,g=r,o=x file1.txt

Another convection using +ve

chmod u+rwx,g+rx,o+w file1.txt

Similarly using -ve removes the permissions   
chmod u-rw,g-w,o-w file1.txt

**Groups:** A group contains multiple users. All the users belonging to a group have the same set of permission access for the file.

Group permissions are owned by the group that owns the file/ directories.

Like Users , groups have some attribute called gid or group identity., that determine thes system resources for the group

**There are range for these id:**

**0: reserved for root**

**1-99:reserved for system and applications**

**100-above: allocated to users groups**

SSH:

This command is used to securely login into a remote machine and execute commands. This provides an encrypted connection between client and server, which ensures that all data transmitted is secure .

Syntax: ssh user@hostname

Ssh ‘username@IP’ (in quotes type command) “ls -l”

Scp:

This is securely copying files between hosts on the network . It encrypts the data being transferred , making it a secured way to share data like ftp.

**Copying from Local to Remote**: scp /path/to/local/file user@hostname:/path/to/remote/directory  
Scp cdac@10.0.2.15:’file path’ ‘destination’

**Copying Directories**: Use the -r option to copy directories recursively.

Example: scp -r /home/vboxuser/mydir cdac@127.0.0.1:/home/cdac/

sudo apt update

sudo apt install openssh-client

sudo apt install openssh-server

Shell Scripting:

A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:

* The Bourne Shell
* The C Shell
* The Korn Shell
* The GNU Bourne-Again Shell

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as **/bin/sh** on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

we create a **cdac.sh** script. Note all the scripts would have the **.sh** extension

Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using th**e shebang construct.**

**#!/bin/sh : This tells the system that the commands that follow are to be executed by the Bourne shell.**

**Program 1:**

Once we have our cdac.sh file created and we've specified the bash shebang on the very first line, we are ready to create our first Hello World bash script.

To do that, open the cdac.sh file again and add the following after the

#!/bin/bash

echo "Hello World!"

You will see a "Hello World" message on the screen.

Another way to run the script would be:

bash cdac.sh

## **Variable Names**

The name of a variable can contain only letters (a to z or A to Z), numbers ( 0 to 9) or the underscore character ( \_).

By convention, Unix shell variables will have their names in UPPERCASE.

## **Defining Variables**

Variables are defined as follows −

variable\_name=variable\_valueq

NAME="Cdac Mumbai"

num= 20

## Variable Types

In shell scripting there are three main types of variables are present. They are –

* Local Variables
* Global Variables or Environment Variables
* Shell Variables or System Variables

## **Local Variable**

A local variable is a special type of variable which has its scope only within a specific function or block of code. Local variables can override the same variable name in the larger scope.

name="CDAC"

echo "Name: $name"

## **Global Variables**

A global variable is a variable with global scope. It is accessible throughout the program. Global variables are declared outside any block of code or function.

## **Shell Variables**

These are special types of variables. They are created and maintained by Linux Shell itself. These variables are required by the shell to function properly They are defined in Capital letters and to see all of them, we can use set / env / printenv command.

### **Environment Variables**

### These are variables that are available system-wide or to any child process initiated by the shell. Environment variables are usually defined in shell configuration files like .bashrc or .bash\_profile.

export PATH="/usr/local/bin:$PATH"

export MY\_VAR="Hello"

### **Readonly Variables**

These are variables that are defined once and cannot be changed or unset.

readonly MY\_VAR="10"

MY\_VAR="20" # This will cause an error

unset My\_VAR # come out from read only mode

| **Sr.No.** | **Variable** | **Description** |
| --- | --- | --- |

| 1 | $0 | The filename of the current script. |
| --- | --- | --- |

| 2 | $n | These variables correspond to the arguments with which a script was invoked. $1 is the first, $2 is the second, and so on. |
| --- | --- | --- |

| 3 | $# | The number of arguments supplied to a script. |
| --- | --- | --- |

| 4 | $\* | All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2. |
| --- | --- | --- |

| 5 | $@ | All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2. |
| --- | --- | --- |

| 6 | $? | The exit status of the last command executed. |
| --- | --- | --- |

| 7 | $$ | The process number of the current shell. For shell scripts, this is the process ID under which they are executing. |
| --- | --- | --- |

| 8 | $! | The process number of the last background command. |
| --- | --- | --- |

**echo "Total Number of Parameters#!/bin/sh**

**echo "File Name: $0"**

**echo "First Parameter : $1"**

**echo "Second Parameter : $2"**

**echo "Quoted Values: $@"**

**echo "Quoted Values: $\*"**

**: $#"**

$\* and $@ are special parameters in shell scripting that represent all the arguments passed to a script or a function. However, they behave differently when used within double quotes.

#### **1. $\***: $\* treats all the arguments as a single string. When referenced within double quotes, all the positional parameters (arguments) are concatenated into a single string, separated by the first character of the IFS (Internal Field Separator), which is usually a space.

* **Behavior**:
  + Without double quotes: $\* expands to a single string containing all the arguments separated by the first character of IFS.
  + With double quotes: "$\*" expands to a single string where all arguments are joined together into one string, separated by the first character of IFS.

**Example**:  
#!/bin/bash

echo "Using \$\*: $\*"\

echo "Using \"\$\*\": \"$\*\""

If the script is run with two arguments arg1 and arg2, like so: ./script.sh arg1 arg2, the output would be:  
  
**Using $\*: arg1 arg2**

**Using "$\*": arg1 arg2**

#### **2. $@:** $@ treats all the arguments as separate individual strings. When referenced within double quotes, each positional parameter (argument) is treated as a separate quoted string.

* **Behavior**:
  + Without double quotes: $@ expands to separate strings for each argument, just like when they were passed.
  + With double quotes: "$@" expands to separate quoted strings for each argument, preserving the arguments as distinct entities.

**#!/bin/bash**

**echo "Using \$@: $@"**

**echo "Using \"\$@\": \"$@\""**

If the script is run with two arguments arg1 and arg2, like so: ./script.sh arg1 arg2, the output would be:  
  
**Using $@: arg1 arg2**

**Using "$@": "arg1" "arg2"**

### **Key DifferencesSummary**

### 

| **Aspect** | **$\*** | **$@** |
| --- | --- | --- |
| **Without Double Quotes** | Arguments are joined as a single string, separated by the first character of IFS. | Arguments are passed as separate strings. |
| **With Double Quotes** | All arguments are joined into a single quoted string. | Each argument is treated as a separate quoted string. |

* Use "$\*" when you want to treat all arguments as a single string.
* Use "$@" when you want to treat each argument as a separate string, which is usually more common and safer, especially when dealing with arguments that might contain spaces or special characters.

## **Exit Status**

The **$?** variable represents the exit status of the previous command.

Exit status is a numerical value returned by every command upon its completion. As a rule, most commands return an exit status of 0 if they were successful, and 1 if they were unsuccessful.

ls /home/vboxuser/root1

echo $?

**Common Exit Status Codes:**

* **0**: Success.
* **1**: General error (catchall for general errors).
* **2**: Misuse of shell builtins (e.g., cd to a non-existent directory).
* **126**: Command invoked cannot execute (permission issue).
* **127**: Command not found (e.g., mistyping a command).
* **128+n**: Fatal error signal n.
* **130**: Script terminated by Ctrl+C.
* **255**: Exit status out of range (exit statuses should be between 0 and 255).

**Program 2:**

Adding our name variable here in the file, with a welcome message. Our file now looks like this:

#!/bin/bash name="CDAC"

echo "Hi there $name"

**Program 3:**

**#!/bin/bash**

**echo "Hello there" $1 # $1 : first parameter echo "Hello there" $2 # $2 : second parameter echo "Hello there" $@ # $@ : all**

**Bash User Input**

With the previous script, we defined a variable, and we output the value of the variable on the screen with the echo $name.

#!/bin/bash

echo "What is your name?"

read name

echo "Hi there $name"

echo "Welcome to CDAC!"

To reduce the code, we could change the first echo statement with the read -p, the read command used with -p flag will print a message before prompting the user for their input:

#!/bin/bash

read -p "What is your name? " name

echo "Hi there $name"

echo "Welcome to CDAC!"

**Slicing:**

Let's review the following example of slicing in a string in Bash:

#!/bin/bash

letters=( "A""B""C""D""E" )

echo ${letters[@]}

This command will print all the elements of an array.

Output:

$ ABCDE

**Program:**

#!/bin/bash

letters=( "A""B""C""D""E" )

b=${letters:0:2}

echo "${b}"

Operators:

In shell scripting, arithmetic operations can be performed using various methods, such as expr, $((...))

**#!/bin/bash**

**# Initialize variables**

**a=10**

**b=3**

**# Addition**

**sum=$((a + b))**

**echo "Addition: $a + $b = $sum"**

**# Subtraction**

**diff=$((a - b))**

**echo "Subtraction: $a - $b = $diff"**

**# Multiplication**

**product=$((a \* b))**

**echo "Multiplication: $a \* $b = $product"**

**# Division**

**quotient=$((a / b))**

**echo "Division: $a / $b = $quotient"**

**# Modulus**

**remainder=$((a % b))**

**echo "Modulus: $a % $b = $remainder"**

**# Exponentiation**

**power=$((a \*\* b))**

**echo "Exponentiation: $a ^ $b = $power"**

**# Increment**

**((a++))**

**echo "Increment: a++ = $a"**

**# Decrement**

**((b--))**

**echo "Decrement: b-- = $b"**

### **Arithmetic Operators in Bash**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | Addition | result=$((a + b)) |
| - | Subtraction | result=$((a - b)) |
| \* | Multiplication | result=$((a \* b)) |
| / | Division | result=$((a / b)) |
| % | Modulus (remainder) | result=$((a % b)) |
| \*\* | Exponentiation | result=$((a \*\* b)) |
| ++ | Increment (add 1) | ((a++)) or ((++a)) |
| -- | Decrement (subtract 1) | ((a--)) or ((--a)) |

Relational operators in shell scripting are used to compare two values or expressions. These operators return a Boolean value (true or false), which can be used in conditional statements like if, while, and until.

### **Common Relational Operators**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| -eq | Equal to | [ $a -eq $b ] |
| -ne | Not equal to | [ $a -ne $b ] |
| -lt | Less than | [ $a -lt $b ] |
| -le | Less than or equal to | [ $a -le $b ] |
| -gt | Greater than | [ $a -gt $b ] |
| -ge | Greater than or equal to | [ $a -ge $b ] |

**#!/bin/bash**

**# Initialize variables**

**a=10**

**b=20**

**# Equal to**

**if [ $a -eq $b ]; then**

**echo "$a is equal to $b"**

**else**

**echo "$a is not equal to $b"**

**fi**

**# Not equal to**

**if [ $a -ne $b ]; then**

**echo "$a is not equal to $b"**

**else**

**echo "$a is equal to $b"**

**fi**

**# Less than**

**if [ $a -lt $b ]; then**

**echo "$a is less than $b"**

**else**

**echo "$a is not less than $b"**

**fi**

**# Less than or equal to**

**if [ $a -le $b ]; then**

**echo "$a is less than or equal to $b"**

**else**

**echo "$a is greater than $b"**

**fi**

**# Greater than**

**if [ $a -gt $b ]; then**

**echo "$a is greater than $b"**

**else**

**echo "$a is not greater than $b"**

**fi**

**# Greater than or equal to**

**if [ $a -ge $b ]; then**

**echo "$a is greater than or equal to $b"**

**else**

**echo "$a is less than $b"**

**fi**

Boolean operators are used to perform logical operations, typically in the context of conditional statements like if, while, and until. The common Boolean operators in Bash are &&, ||, and !.

### **Boolean Operators in Bash**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | Logical AND: True if both conditions are true. | [ $a -gt 0 ] && [ $b -gt 0 ] |
| ` |  | ` |
| ! | Logical NOT: Inverts the condition. | [ ! -f "file.txt" ] |

**AND:**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -gt 5 ] && [ $b -gt 15 ]; then**

**echo "Both conditions are true."**

**else**

**echo "One or both conditions are false."**

**fi**

**OR:**

**#!/bin/bash**

**a=10**

**b=5**

**if [ $a -gt 15 ] || [ $b -gt 3 ]; then**

**echo "At least one condition is true."**

**else**

**echo "Both conditions are false."**

**fi**

**NOT:**

**#!/bin/bash**

**file="example.txt"**

**if [ ! -f "$file" ]; then**

**echo "File does not exist."**

**else**

**echo "File exists."**

**fi**

**String operators are used to manipulate and compare strings. Below is a list of common string operators in Bash, along with examples of how to use them.**

### **Common String Operators in Bash**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| **=** | **Checks if two strings are equal.** | **[ "$a" = "$b" ]** |
| **!=** | **Checks if two strings are not equal.** | **[ "$a" != "$b" ]** |
| **-z** | **Checks if the length of a string is zero (empty).** | **[ -z "$a" ]** |
| **-n** | **Checks if the length of a string is non-zero (not empty).** | **[ -n "$a" ]** |
| **>** | **Checks if a string is greater than another (lexicographically).** | **[ "$a" \> "$b" ]** |
| **<** | **Checks if a string is less than another (lexicographically).** | **[ "$a" \< "$b" ]** |

### **Examples of String Operators**

#### **1. Equality (=)**

**The = operator is used to check if two strings are equal.**

**#!/bin/bash**

**a="hello"**

**b="hello"**

**if [ "$a" = "$b" ]; then**

**echo "Strings are equal."**

**else**

**echo "Strings are not equal."**

**fi**

#### **Inequality (!=)**

**The != operator is used to check if two strings are not equal.**

**#!/bin/bash**

**a="hello"**

**b="world"**

**if [ "$a" != "$b" ]; then**

**echo "Strings are not equal."**

**else**

**echo "Strings are equal."**

**fi**

#### **Empty String Check (-z)**

**The -z operator checks if the length of a string is zero (i.e., the string is empty)**

**#!/bin/bash**

**a=""**

**if [ -z "$a" ]; then**

**echo "String is empty."**

**else**

**echo "String is not empty."**

**fi**

#### **Non-Empty String Check (-n)**

**The -n operator checks if the length of a string is non-zero (i.e., the string is not empty).**

**#!/bin/bash**

**a="hello"**

**if [ -n "$a" ]; then**

**echo "String is not empty."**

**else**

**echo "String is empty."**

**fi**

**File test operators in shell scripting are used to check properties of files and directories, such as whether a file exists, whether it is a regular file or a directory, whether it has specific permissions, and more. These operators are crucial for writing scripts that need to interact with the filesystem.**

### **Common File Test Operators in Bash**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| **-e** | **Checks if a file (or directory) exists.** | **[ -e file.txt ]** |
| **-f** | **Checks if a file exists and is a regular file (not a directory or device).** | **[ -f file.txt ]** |
| **-d** | **Checks if a directory exists.** | **[ -d /path/to/dir ]** |
| **-r** | **Checks if a file is readable.** | **[ -r file.txt ]** |
| **-w** | **Checks if a file is writable.** | **[ -w file.txt ]** |
| **-x** | **Checks if a file is executable.** | **[ -x script.sh ]** |
| **-s** | **Checks if a file exists and is not empty (size is greater than zero).** | **[ -s file.txt ]** |
| **-L** | **Checks if a file is a symbolic link.** | **[ -L symlink ]** |
| **-c** | **Checks if a file is a character special file (device file).** | **[ -c /dev/ttyS0 ]** |
| **-b** | **Checks if a file is a block special file (device file).** | **[ -b /dev/sda ]** |
| **-p** | **Checks if a file is a named pipe (FIFO).** | **[ -p /path/to/fifo ]** |
| **-S** | **Checks if a file is a socket.** | **[ -S /path/to/socket ]** |
| **-u** | **Checks if a file has the setuid bit set.** | **[ -u file.txt ]** |
| **-g** | **Checks if a file has the setgid bit set.** | **[ -g file.txt ]** |
| **-k** | **Checks if a file has the sticky bit set.** | **[ -k /path/to/dir ]** |

#### **Check if a File Exists (-e)**

**The -e operator is used to check if a file or directory exists.**

**#!/bin/bash**

**if [ -e file.txt ]; then**

**echo "file.txt exists."**

**else**

**echo "file.txt does not exist."**

**fi**

**Check if a Regular File Exists (-f)**

**#!/bin/bash**

**if [ -f file.txt ]; then**

**echo "file.txt is a regular file."**

**else**

**echo "file.txt is not a regular file or does not exist."**

**fi**

**Check if a Directory Exists (-d)**

**#!/bin/bash**

**if [ -d /path/to/dir ]; then**

**echo "Directory exists."**

**else**

**echo "Directory does not exist."**

**fi**

**Check if a File is Readable (-r)**

**#!/bin/bash**

**if [ -r file.txt ]; then**

**echo "file.txt is readable."**

**else**

**echo "file.txt is not readable."**

**fi**

**Check if a File is Not Empty (-s)**

**#!/bin/bash**

**if [ -s file.txt ]; then**

**echo "file.txt is not empty."**

**else**

**echo "file.txt is empty or does not exist."**

**fi**

**Bash Conditional Expressions**

In Bash, conditional expressions are used by the [[ compound command and the [built-in commands to test file attributes and perform string and arithmetic comparisons.

To match True condition: [[ ${string1} == ${string2} ]]

True if the strings are not equal.: [[ ${string1} != ${string2} ]]

**Arithmetic operators**

Returns true if the numbers are **equal** : [[ ${arg1} -eq ${arg2} ]]

Returns true if the numbers are **not equal** : [[ ${arg1} -ne ${arg2} ]]

Returns true if arg1 is **less than** arg2: [[ ${arg1} -lt ${arg2} ]]

Returns true if arg1 is **less than or equal** arg2: [[ ${arg1} -le ${arg2} ]]

Returns true if arg1 is **greater than** arg2: [[ ${arg1} -gt ${arg2} ]]

Returns true if arg1 is **greater than or equal** arg2: [[ ${arg1} -ge ${arg2} ]]

AND: [[ test\_case\_1 ]] && [[ test\_case\_2 ]]

OR: [[ test\_case\_1 ]] || [[ test\_case\_2 ]]

**#!/bin/bash**

**p\_pass="CDAC"**

**read -p "Enter your username? " password**

**# Check if the username provided is the admin if [[ "${p\_pass}" == "${password}" ]] ; then**

**echo "You have entered the correct password!"**

**else**

**echo "You have entered the wrong password!"**

**fi**

**Program:**

**#!/bin/bash**

**read -p "Enter a number: " num if [[ $num -gt 0]] ; then**

**echo "The number is positive" elif [[ $num -lt 0]] ; then**

**echo "The number is negative" else**

**echo "The number is 0"**

**fi**

**#!/bin/bash**

**# Prompt the user to enter a number**

**echo "Enter a number between 1 and 3:"**

**read number**

**# Check the value of the number using if, elif, else**

**if [ "$number" -eq 1 ]; then**

**echo "You entered one."**

**elif [ "$number" -eq 2 ]; then**

**echo "You entered two."**

**elif [ "$number" -eq 3 ]; then**

**echo "You entered three."**

**else**

**echo "You entered a number outside the range of 1 to 3."**

**fi**

### **Checking if a Number is Positive, Negative, or Zero**

**#!/bin/bash**

**# Prompt the user to enter a number**

**echo "Enter a number:"**

**read number**

**# Check if the number is positive, negative, or zero**

**if [ "$number" -gt 0 ]; then**

**echo "The number is positive."**

**elif [ "$number" -lt 0 ]; then**

**echo "The number is negative."**

**else**

**echo "The number is zero."**

**fi**

**Checking if a Character is a Vowel or a Consonant**

**#!/bin/bash**

**# Prompt the user to enter a single character**

**echo "Enter a single letter:"**

**read char**

**# Convert character to lowercase to make the check case-insensitive**

**char=$(echo "$char" | tr '[:upper:]' '[:lower:]')**

**# Check if the character is a vowel or consonant**

**if [ "$char" == "a" ] || [ "$char" == "e" ] || [ "$char" == "i" ] || [ "$char" == "o" ] || [ "$char" == "u" ]; then**

**echo "$char is a vowel."**

**elif [[ "$char" =~ [a-zA-Z] ]]; then**

**echo "$char is a consonant."**

**else**

**echo "$char is not a valid letter."**

**fi**

**Switch case statements**

As in other programming languages, you can use a case statement to simplify complex conditionals when there are multiple different choices. So rather than using a few if, and if-else statements, you could use a single case statement.

The Bash case statement syntax looks like this:

case $some\_variable in pattern\_1)

commands

;;

pattern\_2| pattern\_3) commands

;;

\*)

default commands

;;

esac

A quick rundown of the structure:

All case statements start with the case keyword.

On the same line as the case keyword, you need to specify a variable or an expression followed by the in keyword.

After that, you have your case patterns, where you need to use ) to identify the end of the pattern.

You can specify multiple patterns divided by a pipe: |.

After the pattern, you specify the commands that you would like to be executed in case that the pattern matches the variable or the expression that you've specified.

All clauses have to be terminated by adding ;; at the end. You can have a default statement by adding a \* as the pattern.

To close the case statement, use the esac (case typed backwards) keyword.

Here is an example of a Bash case statement:

**#!/bin/bash**

**read -p "Enter the name of your car brand: " car case $car in**

**Tesla)**

**echo -n "${car}'s car factory is in the USA."**

**;;**

**BMW | Mercedes | Audi | Porsche)**

**echo -n "${car}'s car factory is in Germany."**

**;;**

**Toyota | Mazda | Mitsubishi | Subaru)**

**echo -n "${car}'s car factory is in Japan."**

**;;**

**\*)**

**echo -n "${car} is an unknown car brand"**

**;;**

**esac**

### **Days of the Week**

**#!/bin/bash**

**# Prompt the user to enter a day of the week**

**echo "Enter a day of the week (e.g., Monday, Tuesday, etc.):"**

**read day**

**# Convert input to lowercase to make the check case-insensitive**

**day=$(echo "$day" | tr '[:upper:]' '[:lower:]')**

**# Use a case statement to match the input to a specific day**

**case $day in**

**monday)**

**echo "Today is Monday. Start of the workweek!"**

**;;**

**tuesday)**

**echo "Today is Tuesday. Second day of the workweek!"**

**;;**

**wednesday)**

**echo "Today is Wednesday. Midweek already!"**

**;;**

**thursday)**

**echo "Today is Thursday. Almost the weekend!"**

**;;**

**friday)**

**echo "Today is Friday. The weekend is near!"**

**;;**

**saturday)**

**echo "Today is Saturday. Enjoy your weekend!"**

**;;**

**sunday)**

**echo "Today is Sunday. Rest well for the week ahead!"**

**;;**

**\*)**

**echo "That's not a valid day of the week."**

**;;**

**esac**

### **Basic Calculator**

**#!/bin/bash**

**# Prompt the user to enter two numbers**

**echo "Enter the first number:"**

**read num1**

**echo "Enter the second number:"**

**read num2**

**# Prompt the user to choose an operation**

**echo "Choose an operation (+, -, \*, /):"**

**read operation**

**# Use a case statement to perform the chosen operation**

**case $operation in**

**+)**

**result=$((num1 + num2))**

**echo "The result of $num1 + $num2 is: $result"**

**;;**

**-)**

**result=$((num1 - num2))**

**echo "The result of $num1 - $num2 is: $result"**

**;;**

**\\*)**

**result=$((num1 \* num2))**

**echo "The result of $num1 \* $num2 is: $result"**

**;;**

**/)**

**if [ $num2 -ne 0 ]; then**

**result=$((num1 / num2))**

**echo "The result of $num1 / $num2 is: $result"**

**else**

**echo "Error: Division by zero is not allowed."**

**fi**

**;;**

**\*)**

**echo "Invalid operation. Please choose +, -, \*, or /."**

**;;**

**esac**

### **Menu-Driven Program**

**#!/bin/bash**

**while true; do**

**# Display the menu**

**echo "Please choose an option:"**

**echo "1. Display the current date and time"**

**echo "2. List files in the current directory"**

**echo "3. Show the current user"**

**echo "4. Exit"**

**# Read the user's choice**

**read -p "Enter your choice [1-4]: " choice**

**# Use a case statement to perform the selected action**

**case $choice in**

**1)**

**echo "Current date and time: $(date)"**

**;;**

**2)**

**echo "Files in the current directory:"**

**ls**

**;;**

**3)**

**echo "Current user: $USER"**

**;;**

**4)**

**echo "Exiting the program. Goodbye!"**

**break**

**;;**

**\*)**

**echo "Invalid option. Please choose a number between 1 and 4."**

**;;**

**esac**

**echo “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”**

**done**

**For loops**

Here is the structure of a for loop:

for var in ${list} do

your\_commands

done

Program:

**#!/bin/bash users="CDAC Kharghar, Mumbai" for user in ${users}**

**do**

**echo "${user}"**

**done**

**#!/bin/bash**

**for num in {1..10}**

**do**

**echo ${num}**

**done**

### **Iterating Over a List of Items**

**#!/bin/bash**

**fruits=("apple" "banana" "cherry" "date" "elderberry")**

**for fruit in "${fruits[@]}"; do**

**echo "I like $fruit"**

**done**

#### **Iterating Over Files in a Directory**

**#!/bin/bash**

**# Use a for loop to list all files in the current directory**

**for file in \*; do**

**echo "Found file: $file"**

**done**

### **Calculating the Sum of a List of Numbers**

**#!/bin/bash**

**numbers=(10 20 30 40 50)**

**sum=0**

**for number in "${numbers[@]}"; do**

**sum=$((sum + number))**

**done**

**echo "The total sum is: $sum"**

**While loops**

The structure of a while loop is quite similar to the for loop:

while [ your\_condition ]

do

your\_commands

done

Here is an example of a while loop:

**#!/bin/bash**

**counter=1**

**while [[ $counter -le 10 ]]**

**do**

**echo $counter ((counter++))**

**done**

**shell Input/Output Redirections**

Input and output redirection are fundamental concepts in shell scripting that allow you to control the flow of data to and from commands. Here’s an overview of how input/output redirections work in the shell, along with examples.

### **1. Output Redirection**

Output redirection allows you to direct the output of a command to a file or another command instead of the standard output (which is usually the terminal).

#### **Redirecting Standard Output (> and >>)**

**>:** Redirects the output to a file. If the file exists, it will be overwritten. If the file does not exist, it will be created.  
Example:  
 **echo "Hello, World!" > output.txt**

**This command writes "Hello, World!" to output.txt. If output.txt exists, it will be overwritten.**

**>>:** Appends the output to a file. If the file does not exist, it will be created.  
Example:  
  
**echo "Hello again!" >> output.txt**

**This command appends "Hello again!" to output.txt without overwriting the existing content.**

### **2. Input Redirection**

Input redirection allows you to take input for a command from a file instead of the standard input (which is usually the keyboard).

#### **Redirecting Standard Input (<)**

**<:** Redirects the input from a file. **Example:  
  
cat < input.txt**

**This command reads the contents of input.txt and displays them using cat.**

### **3. Redirecting Standard Error**

Standard error redirection is used to direct error messages to a file or another location.

#### Redirecting Standard Error (2> and 2>>)

**2>:** Redirects standard error to a file. If the file exists, it will be overwritten.  
Example:  
 **ls /nonexistent 2> error.log**

This command tries to list a non-existent directory, and the error message is redirected to error.log.

**2>>:** Appends standard error to a file. If the file does not exist, it will be created.  
Example:  
  
**ls /anothernonexistent 2>> error.log**

This command appends the error message to error.log without overwriting the existing content.

### **4. Redirecting Both Standard Output and Error**

Sometimes you may want to redirect both standard output and error to the same file.

#### Redirecting Both Standard Output and Error (&> and 2>&1)

**&>:** Redirects both standard output and standard error to a file.  
Example:  
 **ls /nonexistent &> output\_and\_error.log**

This command redirects both the standard output and error to output\_and\_error.log.

**2>&1:** Redirects standard error to the same place as standard output.  
**Example:  
  
ls /nonexistent > output\_and\_error.log 2>&1**

This command first redirects standard output to output\_and\_error.log, then redirects standard error to the same file.

### **5. Here Document (<<)**

A here document allows you to redirect multiple lines of input to a command.

**Example:**

cat << EOF

This is line 1.

This is line 2.

EOF

This command sends the lines between << EOF and EOF as input to the cat command, which then outputs them.

### **6. Here Strings (<<<)**

**A here string redirects a single string as input to a command.**

**Example:**

grep "hello" <<< "hello world"

This command searches for the string "hello" in the provided string "hello world".

**Function:**

A function is a block of code that you can define once and call multiple times throughout your script. Functions help you to organize and reuse code, making your scripts more modular and easier to maintain.

**#!/bin/bash**

**# Define a function named greet**

**greet() {**

**echo "Hello, $1!"**

**}**

**# Call the function greet with an argument**

**greet "CDAC"**

**greet "MUMBAI"**

**#!/bin/bash**

**# Define a function that adds two numbers**

**add\_numbers() {**

**result=$(( $1 + $2 ))**

**echo "The sum of $1 and $2 is: $result"**

**}**

**# Call the function with two arguments**

**add\_numbers 5 10**

**#!/bin/bash**

**# Define a function that returns the square of a number**

**square() {**

**echo $(( $1 \* $1 ))**

**}**

**# Capture the return value in a variable**

**result=$(square 4)**

**# Print the result**

**echo "The square of 4 is: $result"**

**#!/bin/bash**

**# Define a function that checks if a number is positive, negative, or zero**

**check\_number() {**

**if [ $1 -gt 0 ]; then**

**echo "$1 is positive."**

**elif [ $1 -lt 0 ]; then**

**echo "$1 is negative."**

**else**

**echo "$1 is zero."**

**fi**

**}**

**# Call the function with different arguments**

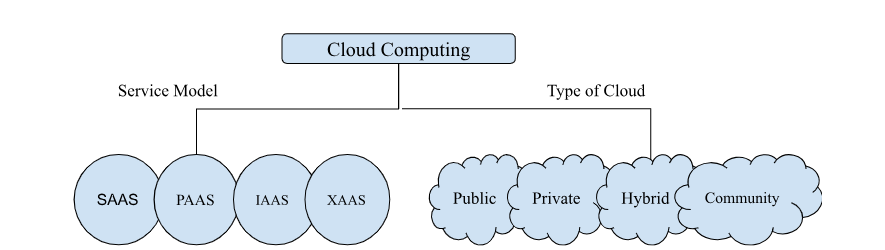
**check\_number 10**

**check\_number -5**

**check\_number 0**

**Cloud Computing**

Cloud computing are services that are provided over the internet or a network which helps you to process the data by using paid or unpaid services. In cloud computing you pay for the services you are going to use (referred to as pay as you go model)



In cloud computing, computing means processing any data or modifying any data whereas cloud stands for network / internet that provides services virtually.

**Data Center:**

**A Data Center** is a facility that is used to store, manage and distribute large amounts of data. This is typically a location / center-point where a large number of computer systems, computing devices, networking equipment, storage devices work together to support processing and storing of data.

**Benefits of Data Center**

1. Storing and Processing of Data   
   A data center is used to store and process large amounts of data which includes files, databases and other digital information.
2. A data center can be used for backup and data recovery, these backups can be very useful during disasters.
3. It allows the business to access computing resources over the internet,
4. We can run applications like e-mail and any other social media platform services on data centers.

To maintain a data center we need cooling, electricity and power backup.

**On Premises** - It is a traditional approach where all the data, hardware are hosted physically on an organization / for an organization   
  
**Off Premises** - Off premises means where the cloud provider provides the services which mean you have to pay for the services you are using. Off premises is virtually located that gives you computing and other services on demand

**Difference between On Premises and Off Premises**

| **On Premises** | **Off Premises** |
| --- | --- |
| We need to buy all the licenses for software and to buy all the hardware | We rent the hardware and software according to our uses |
| It needs a lot of space and security to run and maintain | Here servers in data centers are managed by owners of the data centers |
| Backup is not centralized | Backup is centralized |
| We need people / team to maintain hardware and software | Service providers take care of management of servers and software. |

**Data Center comprises of**

1. **Servers** - They are used to process the request and delivery of the responses over cloud.
2. **Networking Equipment -** These comprises of switches, routers that connects multiple networks to send and receive any data
3. **Server Rack** - It is used to organize servers and another network equipments
4. Cooling Devices and Power Rack

**Data Center provides services under 4 tiers :**   
 **Tier 1** - It provides only power supply   
 **Tier 2** - It provides power supply and cooling.   
 **Tier 3** - It provides cooling, power supply along with maintenance   
 **Tier 4** - It provides cooling, power supply, maintenance, backup and protection of data.

The largest data center in the world is present in **Beijing China** and the largest in India is present in **Greater Noida.**

**Types of Data Centers**

* **Enterprise Data Centers**These kinds of data centers are **owned by a particular organization** and these are **operated by those organizations**. These kinds of data centers can be on premises and off premises, the primary function of these kinds of data centers is secure data and critical application that is essential for the organization.   
    
  Examples - **Amazon, Apple, IBM, Tesla, Microsoft**
* **Cloud Data Centers**

These kinds of data centers are owned and operated by cloud service providers like AWS, Azure, Google Cloud Computing. These data centers deliver cloud computing services **like infrastructure, software to the organization over the network.**

* **CoLocation Data Centers**

These kinds of **data centers are hosted by third-party orgs and are used by an organization.** The **third party organization provides power and cooling** to the data center while the **organization provides server hardware, storage and network.The** company can have equipment located at multiple locations and the company can have their devices maintenance and security.

**Benefits of these kinds of data centers are** 1. It is cheaper than any other data center as few of the resources are rented.   
 2. It can be located at any geographical location.

* **Edge Data Centers**

It is closely **located to end us**er and the IoT devices they serve, Benefits of these kinds of data centers are   
 1. These are located closer to the organization so **network latency is very les**s

* **Hybrid Data Centers**

These types of data centers are a combination of **on premises and off premises data centers.** These are integrated forms of data centers where traditional on premises provides **infrastructure such as servers, storage, network equipment** that is combined with cloud res**ources from one or more cloud service providers**. These kinds of data centers dynamically allocate the load between on premises and off premises.  
  
**We use the cloud to connect to a data center.**

Cloud is a virtual resource while Data centers are physical resources.

It is easier to scale up and down in the cloud as per our requirement whereas it isn’t so feasible with data centers as we need to plan and invest on hardware and software.

# 

# 

# **Cloud Service Models:**

Service models provide you services to process, store and compute data on the remote server. These cloud services are further divided into

**Infrastructure as a Service (IaaS) :**These services provide a set of computing resources such as storage, network, processor that are provided virtually by cloud providers so that users can access and configure according to their uses.   
 In this users rent infrastructure and pay for the services based on their hourly, weekly, monthly usage. It also provides OS servers and database   
**Examples - EC2, Google Compute Engine and Digital Ocean**  
**Benefits of IaaS** Users can modify the architecture as per their requirements, users have complete control over computing resources and they do not have to worry about the maintenance.   
Following are the companies that provide IaaS   
 **AWS, Open Stack, Rack Space, IBM, VMWare**

**Platform as a Service (PaaS) :**   
This provides the platform where developers can develop, test and manage their applications. This service provides an on demand environment for developing any software application. These services are hosted on cloud and users can access these services through a browser.  
 The service provider hosts the hardware and software on their own infrastructure  
Example - **Windows Azure, Google App Engine, Open Swift**  
  
**Benefits of PaaS**As the service provider provides the service (hardware and software) the developer only has to access the services and build the application. It is easier to deploy and application  
  
**Difficulty**To migrate from one service provider to another one is difficult to do.

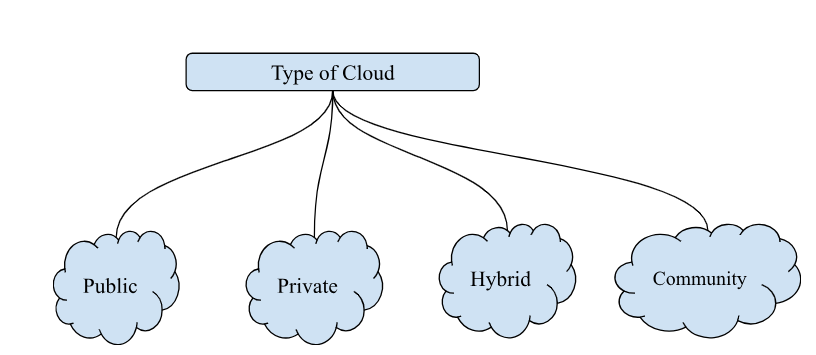
**Software as a Service (SaaS):**   
This provides the user complete software application over the internet, all the infrastructure application tools are located at data centers and managed by service providers. The users don’t have to install any software, they only have to use it.  
  
Example - GMail, GDrive, GDocs, Microsoft Teams   
Vendors - Google Apps, Azure, Dropbox

**SaaS is further divided into two parts :**   
**A. Simple Multi Tenancy**: This means each user will have independent resources that are different from other users   
**B. Fine Grain Multi Latency:** This means each user will get shared resources where the resources are shared among multiple users.  
  
**Benefits of SaaS:**Easy to access application software.  
Application can be accessed from multiple devices and it is low maintenance

1. Anything / Everything as a Service (XaaS)
2. Function as a Service (FaaS)

| **IAAS** | **PAAS** | **SAAS** |
| --- | --- | --- |
| It provides a virtual data center to store information and to provide a platform for testing development deployment. | It provides you virtual platform tools to test and deploy virtual applications. | It provides applications to compute business tasks. |
| It is used by system administrator | It is used by developers. | It is used by end users. |
| It provide infrastructure | It provides infrastructure and platform. | It provides infrastructure platforms and software. |

| **On Premise** | **IAAS** | **PAAS** | **SAAS** |
| --- | --- | --- | --- |
| This is managed by user or an organization | Till OS it is managed by user/organization | Till Data user/ organization manages everything | Everything is managed by vendors |
| Application | Application | Application | Application |
| Data | Data | Data | Data |
| Runtime | Runtime | Runtime | Runtime |
| Middleware | Middleware | Middleware | Middleware |
| OS | OS | OS | OS |
| Virtualization | Virtualization | Virtualization | Virtualization |
| Server | Server | Server | Server |
| Storage | Storage | Storage | Storage |
| Networking | Networking | Networking | Networking |



Cloud is further divided into four parts -

1. **Public**   
    Here resources are open to all and the cloud infrastructure is made available to the general public. The resources are owned by third party   
   **Examples:** EC2, Google App  
     
   **Benefits of Public Cloud**   
   It is easier to scale up and down as per demand  
     
   **Disadvantages**   
   As it is public security breach can happen   
   It is not 100% customized as per user requirement
2. **Private**

They are operated and owned by a single organization which is further divided into two parts   
A. On Premises   
B. Outsourced Private Cloud   
  
**Benefits of Private Cloud:**   
It has high security as only authorized users can only access, these kinds of cloud are usually preferred in the banking sector   
As it is owned by a single organization, it has full control over the resources.  
  
**Disadvantages**  
Only a skilled person can manage and operate these kinds of clouds. Scaling up and down is a bit difficult.

1. **Hybrid**

This kind of cloud is a combination of private and public cloud, which allows the user to share the data between them. These kinds of cloud are partially secure as one of the clouds is public.

This kind of cloud is a combination of 1 public 1 private, 2 or more private clouds, 2 or more public.  
  
The workload between two cloud should be managed efficiently, the performance of hybrid cloud depends on deployment and management of its connection. The link between private and public cloud is done via LAN/VPN  
  
**Benefits of Hybrid Cloud**:   
As it uses combination of public cloud it is easier to scale up and down  
  
**Disadvantages**:   
It is difficult to combine two cloud networks  
Examples : Dedicated interconnection via Google Cloud, Direct Connected via AWS, Express Cloud cia Azure.

1. **Community**

This kind of cloud allows systems and services to be accessible by groups of several organizations / users to share information between them.   
It is owned and managed by many organizations. The infrastructure is shared among different organizations, these kinds of cloud are used in healthcare dept, media dept

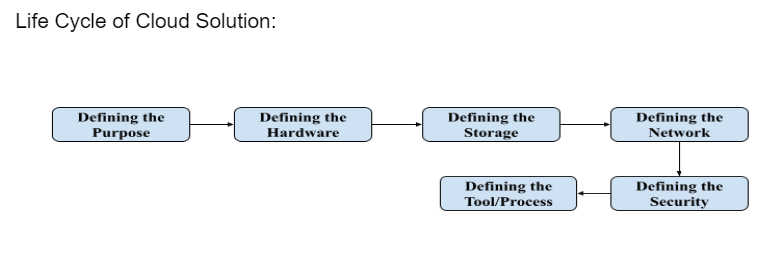
**Benefits of Community**   
1. Maintenance can be shared among the organization   
2. It is less expensive than private cloud   
  
**Disadvantage**

1. It is difficult to distribute the responsibility among the organizations of the same community.

| **Public** | **Private** | **Hybrid** | **Community** |
| --- | --- | --- | --- |
| Host are service providers | Third party are the hosts | Third party are the hosts | Third party are the hosts |
| User are general public in this | Authorized users | Authorized users | Users that belong to same interest/group/community |
| Can be accessed through internet | Internet and VPN | Internet and VPN | Internet and VPN |
| Owners in public are service providers | Organizations are the owners | Organizations are the owners | Group / community are the owners |

Life Cycle of Cloud Solution:

**Defining the Purpose —> Defining the Hardware —> Defining the Storage —--> Defining the Network—--> Defining the Security —--> Defining the Tool / Process**



**Defining the Purpose** : This means to understand the requirement and what type of application or program user / organization needs to build

**Defining the Hardware :** In this stage You need to choose compute resources that provides you right support to scale up and scale down as per the requirement

Examples - **EC2, Lambda, Elastic Container**

**Defining the Storage :** You need to choose right compute services that will provide you right support to store data where you can create backups over the internet   
Examples - **S3, EFSA (Elastic Serverless File System ), Glacier**  
**Defining the Network :** It delivers data and application with low network latency and high speed performance

Examples - **VPC(Virtual Private Cloud), Route 53, Direct Connection**  
**Defining the Security :** We defined security for authorisation and authentication of a user   
Examples - **IAM (Identity and Access Management),**

**Defining the Management Process / Tool :** You can have full control over cloud management by defining the managing tools which monitor AWS Resources, Customer Applications running on the platform.

Examples - **CloudWatch that monitors and manages services provided by cloud platforms.**

AutoScaling - This continually monitors and checks whether the application is running properly or not

# Virtualization:

It is a concept to create a virtual system which uses the host machine’s resources like storage, processor. A single machine can have multiple OS running at once parallelly because of this virtual layer.





**Hypervisor** :  
It is a form of virtual software that is used to allocate resources. It is a hardware virtualization technique that allows you to have multiple OS running on the single host machine at the same time. It is also called as virtual machine manager



Type1: 

This type of hypervisor runs directly on the host machine hardware where it has the direct access to resources, these kinds of hypervisors does not need any server based OS

Type2:

Host OS runs on Host system, these kinds of hypervisor needs host OS or application on the host machine

Examples : VMWare Workstation, Oracle Workbox



**EC2:**

It is a web service that provides secure and resizable compute capacity in the cloud. EC2 allows you to configure the capacity and provides you complete control of computing resources. In this user can scale up and scale down as per their requirements, user can create an environment or instance as per their requirement.

EC2 is a virtual machine that is hosted on AWS

**What is an EC2 instance ?**

An EC2 instance acts as a virtual server in Elastic Cloud Compute (EC2), for running an application on AWS infrastructure.

**The pricing options in EC2 are**

1. **On Demand** - These kinds of pricing are used for short term applications and for unpredictable workload, in this option we pay by hours, minutes and seconds depending upon the type of instance
2. **Reserved** - When you have a fixed workload then you’ll use this kind of pricing option which is further divided into three parts.   
   A. **Standard Reserved Pricing** - This includes up to 70% of discount   
   B. **Convertible Reserved Pricing** - This is up to 50% of discount   
   C. **Scheduled Reserved Pricing** - This does not have any discount

**Spot Instance** - If you urgently need a large amount of computing capacity then this is the best instance you can purchase, unused capacity of up to 90% of discount. The fluctuation of price will also differ depending up on the supply and demand

**Dedicated Instance** - A physical EC2 is dedicated for the use of application, this is one of the most expensive options

**Saving Plans** - You can get up to 72% of discount regardless of instance type . But the user has to commit for the requirement for at least 2-3 years

Basic Terminology used in EC2

1. **Shared Instance**   
     
   A.In this instance model multiple users share the same physical hardware resources such as CPU, memory and storage but they all are logically isolated from each other.

B.Each user's workload runs within its own virtual environment / container, ensuring that they can not directly access other users' data.  
  
 C. This is a multi-user environment which is most cost effective and good for resource utilization.

1. **Dedicated Instance**  
   A.In this kind of instance model, the user has exclusive access to a portion of physical hardware resources. In this unlike shared instance the resources are dedicated to one user  
     
   B.This kind of instances can be used in a scenario where you need high security and high performance application or you need to handle sensitive data with high level of computation power
2. **Dedicated Host**   
     
   This refers to a physical server that is dedicated to a single user / organization.  
   Instead of virtualization the server resources are shared among multiple users, A dedicated host is provided for exclusive access to the entire physical machine.   
   With dedicated hosts, users have the flexibility to deploy multiple VM’s / containers on the same physical server

**Amazon Machine Images (AMI)**

These are pre configured virtual machine images that you can use to launch EC2, you can create and customize your own AMI to save time and efforts in setting up a new instance. There are multiple images that are present

1. **Amazon Linux 2 / AMI 2023** - This is pre configured amazon linux OS
2. **Windows Server AMIs** - These are pre configured images of Microsoft Windows server that are used to launch windows based applications.
3. **Database AMIs** - These are popularly used to database’s such as MySQL, Oracle, Amazon RDS, MongoDB.
4. **Application AMIs** - These are images of applications such as WordPress, Joomla and many others.
5. **Machine Learning AMIs** - These are images of amazon machine learning environments such as SageMaker, TensorFlow, ApacheMXNet and others.
6. **Security AMIs** - These are images of security focused software such as firewalls, intrusion detection systems.

**Instance Types**

AWS EC2 provides you a wide range of instance types to meet your requirements of compute, these instances differ in terms of CPU, Memory, Storage Networking Capacity following are the various types of instances.

1. **General Purpose (M-Series)**   
   These instances provide you with a balanced combination of compute memory and network resources. These are widely used in applications like web servers, small to medium database, deployment and testing   
     
   Example - m5.large, mx.large
2. **Compute Optimizer (C-Series)**These kinds of instances are used for CPU intensive workload, these are good for doing tasks like data analysis, scientific modification and simulation   
     
   Example - c5.large, c48x.large
3. **Memory Optimizer (X and R-Series)**These are used for high memory capacity or for memory intensive applications, these are suitable in memory database, caching applications  
     
   Example - r4x.large, r5.large
4. **Storage Optimizer (R and D-Series)**These are designed for high input,output performance and storage capacity. These are used in data warehousing and big data workload  
     
   Example - i3.large, i3.8xlarge
5. **Accelerated Computing (P and G-Series)**These are used as graphic processing units and are suitable for Machine Learning, Deep Learning and High Performing Computing workloads   
     
   Example - p32x.large
6. **Burstable Performance (T Series)**These are used for high CPU performance and are used for web development and testing environments
7. High Performance Computing (H Series)  
   These are used for high performing test loads like simulating.   
     
   Examples - h1.2xlarge
8. Programmable Gate Array (F1 Series)   
   Examples - f2x.large

**Load Balancer**   
It is a virtual machine that balances your web application load that could be http or http traffic, it balances load of multiple web servers so that no web server is loaded  
  
**Elastic Load Balancing (ELB)**

This automatically manages the workload on your instance and distributes them to other instances. There are three types of load balancers .

1. Application Load Balancer
2. Network Load Balancer
3. Classic Load Balancer

Application Load Balancer

It is used to direct user traffic to public AWS cloud it is best suited for load balancing http traffic, these load balancers are intelligent in sending specific requests to specific servers  
  
Network Load Balancer  
It makes routing decisions at the transport layer and handles millions of requests per second. It is best suited for http traffic when high performance is required  
  
Classic Load Balancer

It routes traffic between client and backend server based on IP address  
  
**Auto Scaling**

It is used to scale up and scale down automatically as and when required. The application available at AWS requires space and load and the auto scaling helps by providing that there are a sufficient number of EC2 instances to handle the load. You can set a limit on EC2 instances such as the number that doesn’t go below threshold.

Autoscaling ensures that your group have sufficient number of servers

**Elastic IPs**

It is a static IPV4 address associated with your AWS account this can be dynamically assigned to any instance or resources within your VPC

**Elastic Block Store (EBS)**

It is a block level storage provided by AWS to use with EC2 instance

***Commands Meaning****:*

**yum update -y :** yum is your package manager that will help you to install packages And -y is a flag for yes prompt.we are running this command to update packages that are created on ec2 to virtual machine.

**yum install httpd -y:** This command will install packages related to http server.

**systemctl start httpd :** This will start the http server on your system.In this command you are instructing the system and service manager of linux to services of http server.this will initiate the web server allowing it to listen to http request.

**systemctl status httpd :** This command used to check the status of http server.This will provide you the information whether the service is running

Properly or not.It will give you the current state ,recent lock and message related to the service.

**systemctl enable httpd :** This command is used to enable http server to start automatically at the time of boot.when we run this command it creates a symbolic link that instructs the system and service manager who start specific services when we start the services.

**cd /var/www/html/ :** We are moving to the html folder.

**AWS STORAGE**

AWS offers a complete range of cloud services to support both application and archival compliance requirements. Select from the objects, files, and block storage services as well as cloud data migration to start designing the foundation of your cloud IT environments.

**Types of storage:**

AWS offers 5 types of storage services such as:

1. Simple Storage Service (S3)

2. Elastic File System (EFS)

3. Elastic Block Store (EBS)

4. Glacier

5. Snowball

**Difference between Object Storage and Block Storage:**

**Block Storage:**

➢ Block storage is suitable for transitional databases, random read/write loads and structured database storage.

➢ Block storage divides the data to be stored in evenly sized blocks called data chunks for instance, a file can be split into evenly sized blocks before it is stored.

➢ Data blocks stored in block storage would not contain metadata. (Data created, data modified, content type etc.)

➢ Block storage only keeps the address (index number) where the data blocks are stored, it does not care what is in that block, just how to retrieve it when required.

**Object Storage:**

➢ Object storage stores the files as a whole and does not divide them.

➢ In object storage an object is: the file/ data itself, its Meta data, object global unique ID. ➢ The object global unique ID is a unique identifier for the object (can be the object name itself) and it must be unique such that it can be retrieved disregarding where it’s physical storage location is.

➢ Object storage cannot be mounted as a drive.

➢ Example of object storage solutions are Dropbox, AWS S3, Facebook.

**Simple Storage Service (S3):**

S3 is Simple Storage Service highly stable, secured cloud based storage provided by AWS. It enables individuals or organizations to store and retrieve objects / files or any unstructured data over the internet.   
  
S3 stores anything in **key-value pairs** where **key** is the name of the **object** and **value** is the **data** made up of sequence of **bytes, version ID** is important for storing versions of the same kind of object.

Metadata stores data about data.

Everything that is stored in S3 is in object format and **the size of object can vary from 0-5 TB**

S3 stores files in buckets which is **similar to folder**, whenever we are working on

S3 we have to consider the following :

1. Universal Namespace: S3 Bucket **namespace should be unique universally**.
2. Whenever you’re uploading a file in **S3 bucket you will receive a http code on successfully uploading the file.**



➢ S3 is a storage for the internet. It has a simple web service interface for simple storing and retrieving of any amount of data, anytime from anywhere on the internet.

➢ S3 is object based storage.

➢ You cannot install operating system on S3.

➢ S3 has a distributed data store architecture where objects are redundantly stored in multiple locations. (minimum 3 locations in same region)

➢ Data is stored in bucket.

➢ A bucket is a flat container of objects.

➢ Maximum capacity of a bucket is 5TB.

➢ You can create folders in your bucket (available through console)

➢ You cannot create nested buckets.

➢ Bucket ownership is non transferrable.

➢ S3 bucket is region specific.

➢ You can have up to 100 buckets per account. (may expand on request)

**S3 Bucket Naming Rules:**

➢ S3 bucket names (keys) are globally unique across all AWS regions.

➢ Bucket names cannot be change after they are created.

➢ If bucket is deleted its name become available again to you or other account to use. ➢ Bucket names must be at least 3 and no more than 63 characters long.

➢ Bucket names are part of URL used to access a bucket.

➢ Bucket name must be a series of one or more labels (xyz bucket)

➢ Bucket names can contain lowercase, numbers and hyphen but cannot use uppercase letters.

➢ Bucket name should not be an IP address.

➢ Each label must start and end with a lowercase letter or a number.

➢ By default buckets and its objects are private, and by default only owner can access the bucket.

**S3 Bucket Sub-Resources:**

Sub-resources of S3 bucket includes:

Lifecycle: to decide on objects' lifecycle management.

Website: to hold configurations related to static websites hosted in S3 buckets. Versioning: keep objects versions as it changes (set updated)

Access Control List: bucket policies

The name is simply two parts: bucket region’s end point / bucket name

Example: for S3 bucket named my bucket in Europe west region is

https://s3-eu-west1.amazonaws.com/mybucket

**S3 Objects:**

➢ An object size stored in an S3 bucket can be 0 byte to 5TB.

➢ Each object is stored and retrieve by unique key. (ID or name)

➢ An object in AWS S3 is uniquely identified and addressed through:

• service endpoint

• bucket name

• object key (name)

• optionally object version

➢ Object stored in a S3 bucket in a region will never leave that region unless you specifically move them to another region or CRR.

➢ A bucket owner can grant cross account permissions to another AWS account (or users in another account) to upload objects.

➢ You can grant S3 bucket / object permission to:

• Individual users

• AWS account

• Make the resource public

• To all authenticate user

**S3 Bucket Versioning:**

➢ Bucket versioning is a S3 bucket sub resource used to protect against accidental object/data deletion or overwrites.

➢ Versioning can also be used for data retention and archive.

➢ Once you enable versioning on a bucket it cannot be disabled however it can be suspended. ➢ When enable, bucket versioning will protect existing and new objects and maintains their versions as they are updated.

➢ Updating objects refers to PUT, POST, COPY, DELETE actions on objects.

➢ When versioning is enable and you try to delete an object a delete marker is placed on the object.

➢ You can still view the object and delete the marker.

➢ If you reconsider deleting the objects you can delete the delete marker and the object will be enable again.

➢ You will be charged for all S3 storage cost for all object versions stored. ➢ You can use versioning with S3 lifecycle policies to delete older version or you can move them to a cheaper S3 storage (Glacier.)

➢ Bucket version state:-

• Enabled

• Suspended

• Un-versioned

➢ Versioning applies to all objects in a bucket and not partially applied.

➢ Object existing before enable versioning will have a version ID or NULL. ➢ If you have a bucket that is already versioned then you suspended versioning existing objects and their versions remain as it is.

➢ However they will not be updated/ version further with future updates while the bucket versioning is suspended.

➢ New objects (uploaded after suspension) they will have a version ID “null” if the same key (name) is used to stone another objects it will override the existing one.

➢ An object deletion in a suspended versioning buckets will only delete the objects with ID “null”.

**S3 Bucket Versioning-MFA Delete:**

➢ Multifactor authentication delete is a versioning capacity that adds another level of security in case your account is compromised.

➢ This adds another layer of security for the following:

• Changing your bucket’s versioning state.

• Permanently deleting on objects version.

➢ MFA delete requires:

• Your security credentials.

• The code displayed on an approved physical or s/w based authentication device.

**S3 Multipart Upload:**

➢ It is used to upload an object in parts.

➢ Parts are uploaded independently and in parallel in any order.

➢ It is recommended for objects sizes of 100MB or larger.

➢ You must use it for objects larger than 5GB.

➢ This is done though S3 multipart upload API.

**Copying S3 Objects:**

➢ The copy operation creates a copy of an objects that is already stored in Amazon S3. ➢ You can create a copy of your object up to 5GB in size a single atomic operation. ➢ However to copy an object greater then 5GB you must use the multipart upload API. ➢ Incur charges if copy to another region.

**Use the copy operation to:**

• Generate additional copies of the subjects.

• Renaming object (copy to a new name)

• Changing the copy’s storage class or encrypt it at rest.

• Move object across AWS location/region.

• Change object metadata.

**STORAGE CLASSES OF AMAZON S3:**

There are 6 types of storage classes of Amazon S3 is available such as:

1. Amazon S3 Standard

2. Amazon S3 Glacier Deep Archive

3. Amazon Glacier

4. Amazon S3 Standard Infrequent Access

5. Amazon S3 one-zone-IA

6. Amazon S3 Intelligent Tiering

**1. Amazon S3 Standard:**

➢ S3 standard offers high durability, availability and performance object storage for frequently accessed data.

➢ Durability is 99.999999999%.

➢ Designed for 99.99% availability over a given year.

➢ Supports SSL for data in transit and encryption of data at rest.

➢ The storage cost for the object is fairly high but there is very less charge for accessing the objects.

➢ Largest object that can be uploaded in a single PUT in 5GB.

**2. Amazon S3 IA (standard):**

➢ S3-IA is for data that is accessed less frequently but requires rapid access when needed.

➢ The storage cost is much cheaper than S3-standard almost half the price, but you are charged more heavily for accessing your objects.

➢ Durability is 99.999999999%.

➢ Resilient against events that impact an entire AZ.

➢ Availability is 99.9% in a year.

➢ Supports SSL for data in transit and encryption of data at rest.

➢ Data that is deleted from S3-IA within 30 days will be charged for a full 30 days. ➢ Backed with the Amazon S3 service level agreement for availability.

**3. Amazon S3 Intelligent Tiering:**

➢ The S3 intelligent tiering storage class is designed to optimize cost by automatically moving data to the most cost effective access tier.

➢ It works by storing objects in two access tiers.

➢ If an object in the frequent access tier is accessed it is automatically moved back to the frequent access tier.

➢ There is no retrieval fees when using the S3 intelligent tiering storage class and no additional tering fees when objects are moved between access tiers.

➢ Same low latency and high performance of S3 standard.

➢ Objects less than 128kb cannot move to IA.

➢ Durability is99.999999999%.

➢ Availability is 99.9%.

**4. Amazon One-Zone IA**

➢ S3 one zone IA is for data that is accessed less frequently but requires rapid access when needed.

➢ Data store is single AZ.

➢ Ideal for those who want lower cost option of IA data.

➢ It is good choice for storing secondary backup copies of on-premise data of easily re-creatable data.

➢ You can use S3 lifecycle policies.

➢ Durability is 99.999999999%.

➢ Availability is 99.5%.

➢ Because S3 one zone IA stores data in a single AZ, data stored in this storage class will be lost in the event of AZ destruction.

**5. Amazon S3 Glacier:**

➢ S3 glacier is a secure, durable, low cost storage class for data archiving. ➢ To keep cost low yet suitable for varying needs S3 glacier provides three retrieval options that ranges from a few minutes to hours.

➢ You can upload object directly to glacier or use lifecycle policies.

➢ Durability is 99.999999999%.

➢ Data is resilient in the event of one entire AZ destruction.

➢ Supports SSL for data in transit and encryption data at rest.

➢ You can retrieve 10GB of your amazon S3 glacier data per month for free with free tier account.

**6. Amazon S3 Glacier Deep Archive:**

➢ S3 glacier deep archive is amazon S3 cheapest storage.

➢ Design to retain data for long period even if for 10 years.

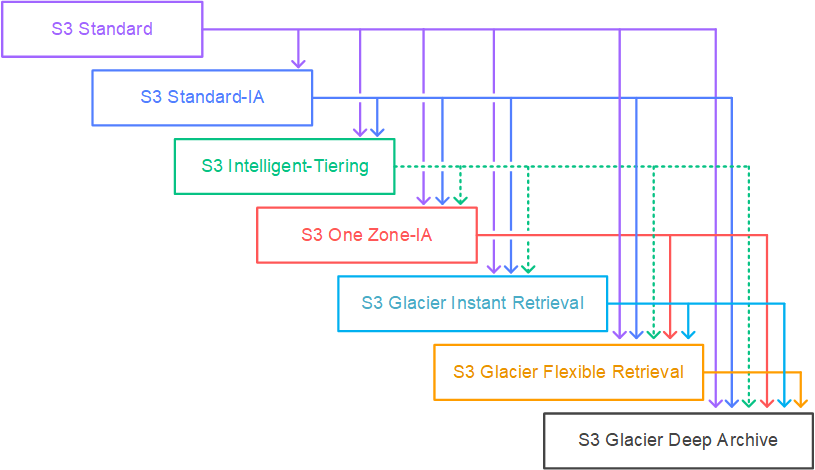
➢ All objects stored in S3 glacier deep archive are replicated and stored across at least at three geographically AZ.

➢ Durability is 99.999999999%.

➢ Ideal alternative to magnetic tape libraries.

➢ Retrieval time within 12 hours.

➢ Storage cost is up to 75% less than for the existing S3 glacier storage class. ➢ Availability is 99.9%.

****

IAM: It stands for identity and access management this refers to policies, processes and technology that are used to manage identity and access the resources and service within the organization.this is designed to ensure that only authorized users or groups can access the sensitive data, application and platform.This help us to prevent unauthorized access and misuse of any data and services.This manages the user and their level of access to the aws console and the resources.

The permission are further divided into four parts:

1. **Centralize** :this gives you centralize control over aws accounts.
2. **Access** : This give you shared access
3. **Permission :**This gives you permissions in which the user will have different level access. This means in the same organization different users will have different levels of access.
4. **Identity federation** :This enables the user to login using their credential store in activity directory ,facebook,linkedin,google.

* Multi factor authentication:user is granted access only after successful completion of multiple independent authentication.for example: user provides username and password its work as one set of authentication and second level authentication is performed by google authentication where token is generated as password.
* Temporary access:It also provides temporary access to user or devices and service .For example: if developing a mobile or web based application you can configure the user to have temporary access to resources within the account.
* Password rotation policy : this allows you to set up your own password .
* Key terminology:

1. User: user is refer as end user who are login in aws console .They can also interact with aws by running api command.
2. Groups: Groups are the collection of users with a common set of permission.

**Roles:**

We can create a set of roles we can assign to the application or service to give access to resources.

**Policies:** they are made of documents called a policy document.these documents are in format that is called json.they give permission to which user role will access which resources.

**Iam policies simulator:** this checks for authentic users and checks where the authorized user is running the resources.